Neural Network Compression for Noisy Storage Devices

BERIVAN ISIK, KRISTY CHOI, XIN ZHENG, TSACHY WEISSMAN, STEFANO ERMON, and H.-S. PHILIP WONG, Stanford University, USA
ARMIN ALAGHI, Reality Labs Research, Meta, USA

Compression and efficient storage of **neural network (NN)** parameters is critical for applications that run on resource-constrained devices. Despite the significant progress in NN model compression, there has been considerably less investigation in the actual *physical* storage of NN parameters. Conventionally, model compression and physical storage are decoupled, as digital storage media with **error-correcting codes (ECCs)** provide robust error-free storage. However, this decoupled approach is inefficient as it ignores the overparameterization present in most NNs and forces the memory device to allocate the same amount of resources to every bit of information regardless of its importance. In this work, we investigate analog memory devices as an alternative to digital media – one that naturally provides a way to add more protection for significant bits unlike its counterpart, but is noisy and may compromise the stored model's performance if used naively. We develop a variety of robust coding strategies for NN weight storage on analog devices, and propose an approach to jointly optimize model compression and memory resource allocation. We then demonstrate the efficacy of our approach on models trained on MNIST, CIFAR-10, and ImageNet datasets for existing compression techniques. Compared to conventional error-free digital storage, our method reduces the memory footprint by up to one order of magnitude, without significantly compromising the stored model's accuracy.

CCS Concepts: • **Computer systems organization → Embedded hardware**; • **Hardware → Memory and dense storage;**

Additional Key Words and Phrases: Neural networks, robustness, compression, analog storage, PCM

ACM Reference format:
Berivan Isik, Kristy Choi, Xin Zheng, Tsachy Weissman, Stefano Ermon, H.-S. Philip Wong, and Armin Alaghi. 2023. Neural Network Compression for Noisy Storage Devices. *ACM Trans. Embedd. Comput. Syst.* 22, 3, Article 58 (May 2023), 29 pages.
https://doi.org/10.1145/3588436

BI is supported by the Stanford Graduate Fellowship and a Meta research award. KC is supported by the NSF GRFP, Stanford Graduate Fellowship, and Two Sigma Diversity Ph.D. Fellowship. The Stanford authors' work was supported by NSF (#1651565, #1522054, #1733686), ONR (N00014-19-1-2145), AFOSR (FA9550-19-1-0024), ARO (W911NF2110125), and Amazon AWS.

Authors' addresses: B. Isik and T. Weissman, Stanford University, 350 Jane Stanford Way, Stanford, CA 94305, US; emails: berivan.isik@stanford.edu, tsachy@stanford.edu; K. Choi and S. Ermon, Stanford University, 353 Jane Stanford Way, Stanford, CA 94305, US; emails: kechoi@stanford.edu, ermon@cs.stanford.edu; X. Zheng and H. S. Philip Wong, Stanford University, 330 Jane Stanford Way, Stanford, CA 94305, US; emails: xzheng3@stanford.edu, hspwong@stanford.edu; A. Alaghi, Meta Reality Labs, 9845 Willows Rd NE, Redmond, WA 98052, US; email: alaghi@meta.com.

1539-9087/2023/05-ART58 $15.00
https://doi.org/10.1145/3588436

1 INTRODUCTION

The rapidly growing size of deep neural networks presents new challenges in their storage, computation, and power consumption for deployment in resource-constrained devices [16, 46]. This makes it crucial to compress and efficiently store NN parameters. The most commonly used approach is to separate the problem of model compression from physical storage. Reliable digital storage media, fortified by **error correcting codes (ECCs)**, provide nearly error-free storage to users – this allows researchers to develop model compression techniques independently from the precise characteristics of the devices used to store the compressed weights [9, 11]. Meanwhile, memory designers strive to create efficient storage by hiding such physical details from users.

Although the decoupled approach enables isolated investigation of model compression and simplifies the problem, it misses the opportunity to exploit the full capabilities of the storage device. With no context from data, memory systems dedicate the same amount of resources to each bit of stored information. This is suboptimal as NNs tend to exhibit a considerable amount of redundancy in their parameterization [12, 76]. To address this shortcoming, we investigate the joint optimization of NN model compression and physical storage – specifically, we perform model compression with the additional knowledge of the memory's physical characteristics. This allows us to dedicate more resources to important bits of data, while relaxing the resources on less valuable bits.

This joint optimization scheme, however, is cumbersome to implement in practice on digital storage media due to the device's physical characteristics (Sections 2.1 and 2.2). We instead turn to analog technology – in particular, **phase-change memory (PCM)** – as a more feasible alternative [42, 53]. Recent studies have demonstrated the promise of end-to-end analog memory systems for storing analog data, such as NN weights, as they have the potential to reach higher storage capacities than digital systems with a significantly lower coding complexity [71–73].

Yet despite their advantages, analog storage devices are noisy and may corrupt the written input values. This presents several key challenges for the compression task. First, the noise characteristic of such memories is a non-linear function of the input value written onto the cell. Second, slight perturbations of the NN weights from the memory cell may cause the network's performance to plummet [1], which is unaccounted for in most NN compression techniques. Thus our objective is to not only minimize the number of memory cells used to store the given NN model (a standard metric named *storage density*), but also preserve the compressed weights' predictive performance.

Motivated by the above challenges, we draw inspiration from classical information theory and coding theory to develop a framework for encoding and decoding NN parameters to be stored on analog devices [62]. In particular, our method: (i) leverages existing compression techniques such as pruning [25, 27] and **knowledge distillation (KD)** [33, 57, 69] to learn a compressed representation of the NN weights; and (ii) utilizes various coding strategies to ensure robustness of the compressed network against storage noise.[1] To the best of our knowledge, this is the first work on NN compression for analog storage devices with *realistic* noise characteristics, unlike previous works that only investigate white Gaussian noise models [22, 74].

In summary, the contributions of our work are as follows:

(1) We develop a variety of strategies to mitigate the effect of noise and preserve the NN performance on PCM storage devices.
(2) We present methods to combine these strategies with existing model compression techniques.

[1]This joint approach is fundamentally different from the problem of preserving the utility of the lossily compressed noisy data [39] or vice-versa – noise-corrupted compressed data.

Empirically, we evaluate the efficacy of our methods on classification tasks with models trained on MNIST, CIFAR-10, and ImageNet datasets and regression tasks with the **Neural Radiance Field (NeRF)** model [52]; and show that storage density can be increased by 18 times on average compared to conventional error-free digital storage.

2 PRELIMINARIES

2.1 Analog Storage and Phase Change Memory (PCM)

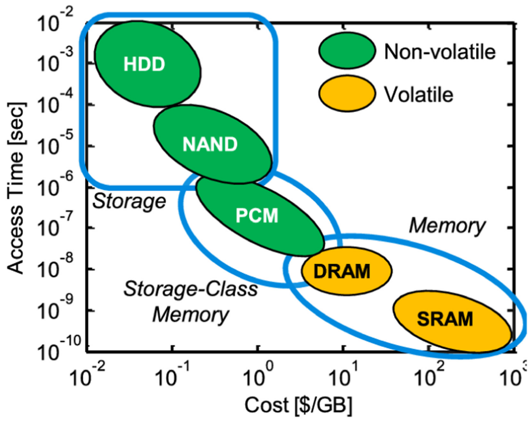
Most memory technologies utilize continuous physical values (e.g., charge) as a means of data storage. The full continuous storage range is often divided into intervals and used to store discrete values. One extreme that maximizes the device's performance is to allow only two values (high and low) to be written to each memory cell. With this approach, one bit of information can be stored in a memory cell. Storing more values allows more bits to be stored per cell, but also increases the chance of reading an incorrect value from the memory – this represents a natural trade-off between memory density and probability of error. Therefore, storage devices often use error correcting codes (ECCs) in practice to protect data from such memory errors.

Although digital storage (i.e., storing discrete values) is the dominant paradigm in physical memory technology, it is still possible to store continuous values in memory cells, and read them back as continuous values, albeit with noise. This approach, also known as *analog storage*, has regained attention recently [72] with the emergence of different **non-volatile memory (NVM)** technologies, such as PCM. NVMs not only retain the stored information even when the power supply is off, but also allow efficient storage of multiple bits per cell. In the extreme case, NVMs can store continuous values [67], making them more efficient than their digital counterparts that require discrete inputs [71, 73]. Among the various NVM technologies, our work focuses on **phase-change memory (PCM)** technology because it: (i) has faster read/write speed and higher endurance than its competitors; and (ii) can enhance chip performance by reducing the cost of data movement thanks to its on-chip integration.

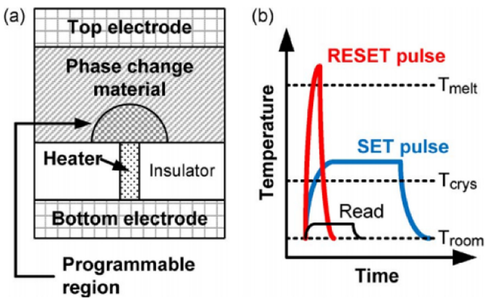
A major advantage of the PCM memory device over its (digital) competitors is its relatively low cost as compared to its access time (read/write time). Commonly used memory devices such as **SRAM (static random access memory)** and **DRAM (dynamic random access memory)** have short access time and high endurance, and are thus used as the cache and main memory in a wide range of technologies. Storage devices such as **NOT-AND (NAND)** flash, on the other hand, provide low-cost, high density, and non-volatile storage, but their long access time makes them unsuitable to use as memory devices. As shown in Figure 1(a), PCM successfully bridges the gap between memory and storage devices. Such advantages of the PCM device allows for utilizing both memory and storage on the main chip, which, when coupled with its high endurance, makes it a compelling alternative to conventional memories such as NAND flash and DRAM. This is especially true for applications requiring NN inference, where both short access time and non-volatile storage are crucial for real-time deployment.

2.1.1 *PCM Basics: Physics Mechanism, Cell Design and Analog Storage Capability.*

After a significant amount of research in both academia and industry, the PCM array has entered the market as a breakthrough technology bridging the gap between memory and storage [24]. A common PCM device consists of a phase change material (e.g., GeSbTe) inserted between the top electrode and bottom electrode as in Figure 1(b)-left. The information in PCM is stored by utilizing the resistivity difference between the low-conductive amorphous state and high-conductive poly-crystalline states of the phase change material. In the following paragraphs, we briefly explain the physical mechanism by which reading and writing take place on a PCM array.



(a)



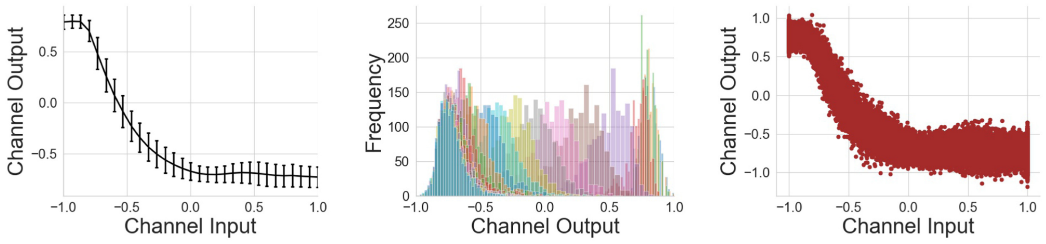
(b)

Fig. 1. (a) Figure from [24] comparing the access time (read/write time) of various memories versus cost. Significant space has opened between conventional digital memories, namely NAND flash and DRAM, in the memory hierarchy. PCM can fill this gap and further complement the memory hierarchy. (b) The cross-section schematic of PCM cell (left) and RESET and SET pulses are used to program the PCM cell with different temperature (right). Read pulse is used to read the resistance of PCM cell. Figure from [67].

To write onto the device, electrical pulses are used to generate phase transformation through joule heating (Figure 1(b)-right). The fresh PCM device is usually in a high-conductive poly-crystalline state. A fast high-temperature (above melting temperature) pulse (RESET pulse) can be used to melt and quench the programming region into low-conductive amorphous states. A longer medium-temperature (above crystallization temperature) pulse, i.e., SET pulse, can then be used to crystallize the programming region back to high-conductive poly-crystalline state. To read from the cell, a smaller pulse is used to measure the resistance of the cell without changing the cell states. In particular, the cell state is measured by reading the cell resistance when applying a small bias (read pulse), whose amplitude is small enough not to disturb the cell. The cell resistance, interestingly, can be continuously tuned as it is decided by the ratio between amorphous region and polycrystalline region. As a result, the PCM device enables analog storage as the cell resistance is an analog value that is determined by the ratio of the two (amorphous and crystalline) phase regions. This property will potentially *increase the storage density of PCM* by storing more than 1 bit per cell.

2.2 Our Setup

2.2.1 Analog-Storage with 1T1R PCM Array: Measurement Details. To simulate realistic storage, we utilize measurements collected from physical experiments on PCM arrays [68] of 1 mega 1T1R cells [68]. We use a simple analog programming strategy by first resetting the device to a low-conductive initial state and then setting it with 31 different pulse amplitudes (input levels) and more than 1,000 cells for each pulse amplitude. Although this simple programming strategy yields higher noise levels than more complex strategies, the write speed is fast. The SET pulse amplitude is controlled with a control transistor connected in series to the PCM device (1T1R structure). Figure 2(a) shows the 1-standard deviation error bar from cell-to-cell variation for each input level, where both the mean and standard deviation of the output (programmed resistance in log scale) exhibits a non-linear relationship with the input. Figure 2(b) is the histogram of output distributions corresponding to the 31 input levels, which shows that the output is roughly



(a) 1-sigma error bar plot. (b) Channel output distribution. (c) Interpolated measurement data.

Fig. 2. Characteristics of the channel noise in a PCM cell. (a) 1-standard deviation error bars for each input level, where both the mean and variation of the output have a nonlinear relationship to the input value. (b) Distribution of output values corresponding to 31 distinct input values into the PCM array. (c) Characteristics of the channel noise in a PCM cell, which is dependent on the input value. Each point corresponds to a possible cell output for a given input.

Gaussian distributed *conditioned on the input level*. The channel response to the input values in between measurement points is then interpolated in order to construct the *differentiable* continuous analog channel model used in this work as shown in Figure 2(c). The differentiability of the model allows for an end-to-end learning scheme (to be discussed in Section 3.3, see Figure 6). Figure 2(c) illustrates the PCM model as a *noisy channel*.

Each point in Figure 2(c) corresponds to a possible read (output) value for a given write (input) value. For simplicity, we linearly map both the inputs and outputs to be within range $[-1, 1]$. Our realistic PCM model has different noise mean and **standard deviation (std)** for each input, which improves over prior works that only investigate white Gaussian noise [22, 74] when storing NN models on noisy storage.

2.2.2 Baselines and Key Assumptions on PCM Storage Modes. In this section, we establish our two digital storage baselines, as well as the set of assumptions used for our approach. For a fair comparison, we use the same memory device (PCM arrays) in *both digital and analog modes* (Section 2.1) to mitigate confounding factors in the hardware technology.

Ideal baseline for PCM digital storage. In classical information theory, the maximum amount of data that can be reliably transmitted across a noisy channel (or as in our case, reliably stored into memory cells) with arbitrarily low error is called the *channel capacity* [62]. In the case of the PCM noise model shown in Figure 2(c), we derive the channel capacity as 2 bits per cell as in [21], and set this as our *ideal* baseline for digital storage. If we assume 32-bit floating point number representation for each NN weight, then this channel capacity implies that the weight can be reliably stored using 16 digital memory cells. Later in Section 4, we also consider less precise representations through quantization, which will serve as our stronger baselines.

Practical baseline for PCM digital storage. Next, we show a more *realistic* baseline for PCM digital storage that is representative of their real-world usage. In practice, various ECCs add extra bits to the data by trading off the overhead coming from additional bits to be stored with the overall reduction in **bit error rate (BER)**. This naturally leads to a rate smaller than the theoretical capacity. We estimate the practically achievable rate of PCM digital storage by multiplying the channel capacity of PCM device (2 bits per cell) by a factor α, where $\alpha < 1$ represents the capacity loss factor (overhead) of practical implementations of **Low Density Parity Check (LDPC)** codes – a commonly used ECC in the industry. Specifically, we extract $\alpha = 0.9$ from Figure 6 of [65] that presents LDPC overhead in an **additive white Gaussian (AWGN)** channel. The practically achievable rate for storing digital data on PCMs would then be 1.8 bits per cell, smaller than the theoretical limit (2 bits per cell). Therefore, a single NN weight (in 32-bit floating point number representation) for this baseline can be reliably stored in 18 digital memory cells. We note that the achieved BER under this assumption is $1e^{-7}$, which is still much larger than the target BER $1e^{-15}$ (digital storage industry standard for acceptable BER). Therefore, 18 cells is actually an optimistic value for the baseline (since the baseline would realistically require more cells in practice), which implies that it is a pessimistic assumption for our method. Nevertheless, we set this as our *realistic* baseline.

PCM analog storage. In our work, we use PCM cells as *analog* storage devices by storing single-precision floating point data into the device without separating the information into bits. We assume that the read/write circuitry of the PCM has a precision equivalent to that of a single-precision (32-bit) floating point data. This means that a floating point number can be directly written to a memory cell, and the data read back is also a 32-bit float, albeit with the added PCM noise. Although a naive application of PCM analog storage would require more than 1K memory cells to store a single NN weight reliably (while eliminating the effect of the PCM noise), we demonstrate that we can drastically reduce this number to outperform all digital baselines (see Tables 1–6).

2.2.3 Additional Assumptions. In addition to the major assumptions above, we outline a few additional assumptions that are less critical. Our work only focuses on weight storage on the PCM device – that is, we assume the chip will not be used for training purposes, so there is no need to store gradients onto the cell. We also assume that the pretrained NN activations will be stored in local caches. If such caches are not sufficient for activation storage, then we will have to pay the costly off-chip storage, because PCMs are not efficient for writing purposes (read is cheap; write is expensive).

2.2.4 Model Limitations. The PCM model used in this work does not consider second-order effects such as device aging and process variation. The latter can be corrected by adding a compensation term to our error model, once wafer-level variation data becomes available to us. Furthermore, our method does not consider cell value tuning because its effect was not available in our PCM model. Our methodology will not be affected by cell value tuning as its effect will only show up as a new and possibly better error model.

2.3 Problem Statement and Notation

To formalize the joint compression problem, we consider a supervised learning setup where $x \in \mathcal{X} \subseteq \mathbb{R}^d$ is the input variable, and $y \in \mathcal{Y} = \{1, \ldots, k\}$ is the label. We assume access to samples $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ drawn from an underlying (unknown) joint distribution $p_{\text{data}}(x, y)$, which are used to train an NN predictor $f_w : \mathcal{X} \to \mathcal{Y}$. This network, parameterized by the weights $w \in \mathcal{W}$ to be compressed and stored on analog storage devices (where $\mathcal{W}$ denotes the parameter space of NNs), indexes a conditional distribution $p_w(y|x)$ over the labels y given the inputs x.

The NN weights w will be exposed to some input dependent device noise $\epsilon(w)$ when written to the analog storage device, yielding noisy versions of the weights $\hat{w} = w + \epsilon(w)$. In our experiments, we find that PCM noise dramatically hurts classification performance – as a concrete example, the test accuracy of a ResNet-18 model trained on CIFAR-10 drops from 95.10% (using w) to 10% (using $\hat{w}$) after the weights are corrupted (via naive storage on analog PCM). Thus, to preserve NN performance even after it is stored on the analog device, we explore various strategies $g : \mathcal{W} \to \mathcal{W}$ for designing *reconstructions* of the perturbed weights $g(\hat{w})$ such that the resulting distribution over the output labels $p_{g(\hat{w})}(y|x)$ is close to that of the original network $p_w(y|x)$.

We note that this notion of "closeness" between the original weights w and the reconstructed weights $g(\hat{w})$ has several interpretations. In Section 3.1.1, we explore methods to minimize the distance between w and $g(\hat{w})$ in Euclidean space:

$$\min_g \|g(\hat{w}) - w\|_2 .$$

In Sections 3.1.2–3.3, we study ways to minimize the **Kullback-Leibler (KL)** divergence between these two output distributions:

$$\min_g \mathbb{E}_{x\sim p_{\text{data}}(x)}[D_{KL}(p_w(y|x)||p_{g(\hat{w})}(y|x))] \tag{1}$$

where we *learn* the appropriate transformation $g(\cdot)$.

3 ROBUST NEURAL NETWORK COMPRESSION

We develop several novel methods to provide NNs robustness against storage noise while also minimizing the storage density. In Section 3.1, we describe several robust coding strategies for NN weights to be applied post-training. We exploit the sparsity and sensitivity of NN weights to make our strategies more efficient. In Section 3.2, we propose robust training and robust distillation methods that simultaneously train the NN model to perform well on the downstream task and be robust to storage noise. We achieve this by regularizing the loss function with KL divergence in Equation (1). Finally, in Section 3.3, we introduce a probabilistic end-to-end approach to optimize compression and robustness (against storage noise) of the NN model.

3.1 Robust Coding Strategies

In this section, we devise several novel coding strategies for $g(\cdot)$ that can be applied to the model post-training to mitigate the effect of perturbations in the weights. For each strategy, we require a pre-mapping process to remove the input dependence in the mean of the PCM response in Figure 2(c).

Pre-mapping: Let C represent the PCM channel. The mean function of C, denoted as $\mu : \mathcal{X} \to \mathbb{R}$, in Figure 2(c) can be learned via a k-nearest neighbor classifier on the channel response measurements. We can also learn an inverse function $h = \mu^{-1}$ using the measurement data and use it to remove the input dependence in the mean. More precisely, we pre-map the data with h prior to channel usage, which yields an identity function with zero-mean noise $\phi = C \circ h$ (Figure 3), i.e., $\phi(x) = x + \epsilon_0(x)$ where $\epsilon_0(x)$ is a zero-mean noise with input dependent std $\sigma(x)$ due to the PCM channel. Thus, the relationship between input weights w_{in} to be stored and output weights w_{out} to be read is:

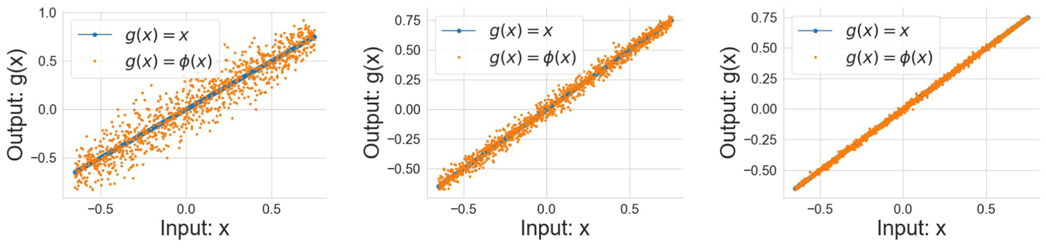
$$w_{out} = \frac{\phi(\alpha \cdot w_{in} - \beta) + \beta}{\alpha},$$

where α and β are scale and shift factors, respectively. Since the noise is zero-mean, we have:

$$w_{out} = \frac{\alpha \cdot w_{in} - \beta + \epsilon_0(w_{in}) + \beta}{\alpha} = w_{in} + \frac{\epsilon_0(w_{in})}{\alpha}.$$

If we use the noisy channel ϕ r times (i.e., store the same weight on r "independent" cells and average over the outputs – much like repetition codes in coding theory), the relationship between w_{in} and w_{out} becomes:

$$\begin{aligned} w_{out} &= \frac{1}{r}\sum_{i=1}^{r}\left(w_{in} + \frac{\epsilon_{0,i}(w_{in})}{\alpha}\right) \\ &= w_{in} + \frac{\frac{1}{r}\sum_{i=1}^{r}\epsilon_{0,i}(w_{in})}{\alpha} \end{aligned} \tag{2}$$



(a) 1 memory cell (b) 10 memory cells (c) 100 memory cells

Fig. 3. Behavior of the inverted channel $\phi = C \circ f$ when outputs are read from an average over 1, 10, and 100 memory cells, respectively.

Let us define a new random variable $\tilde{\epsilon}(w_{in}, r) = \frac{1}{r}\sum_{i=1}^{r} \epsilon_{0,i}(w_{in})$. Notice that the standard deviation of $\tilde{\epsilon}(w_{in}, r)$ is $\frac{\sigma(w_{in})}{\sqrt{r}}$. Then the standard deviation of $\hat{\epsilon}(w_{in}, r, \alpha) = \frac{\tilde{\epsilon}((w_{in}, r))}{\alpha}$ in Equation (2) is given by: $\frac{\sigma(w_{in})}{\alpha\sqrt{r}}$. More precisely, we have:

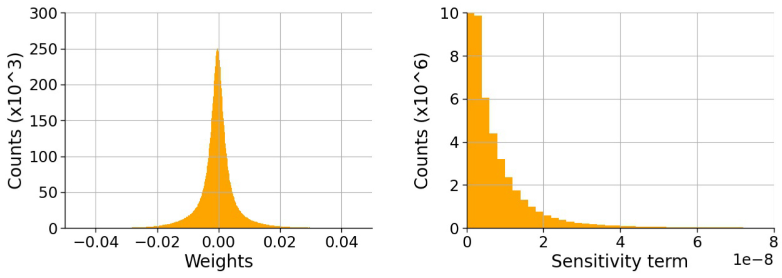
$$w_{out} = w_{in} + \hat{\epsilon}(w_{in}, r, \alpha)$$

where the standard deviation of $\hat{\epsilon}(w_{in}, r, \alpha)$ is $\frac{\sigma(w_{in})}{\alpha\sqrt{r}}$. This provides us two tools to protect the NN weights against zero-mean noise.

(Method #1) Increase the number of channel usage r (number of PCM cells per weight).
(Method #2) Increase the scale factor α under the condition that $\alpha \cdot w_{in}$ satisfies the cell input range limitation ($|\alpha \cdot w_{in} - \beta| \leq 1$).

For the first method, we observe in Figure 3 that the response becomes less noisy as we increase the number of PCM cells used (r). However, we desire to keep r at a minimum for an efficient storage. The second method allows us to make use of the full analog range by scaling the weights. This is particularly useful when storing weights with small magnitude. But we cannot increase α without bound because of the device constraint, since we must satisfy $-1 \leq \alpha \cdot w_{in} - \beta \leq 1$ (in practice, this constraint is $-0.65 \leq \alpha \cdot w_{in} - \beta \leq 0.75$ since the remaining portion of the response is non-invertible, and therefore unusable for our purposes). Such limitations leave more to be desired for a general-purpose robust coding strategy for NN weights.

3.1.1 Sparsity-Driven Protection. Next, we leverage the observation that the weights of a fully-trained NN tend to be sparse [40] (Figure 4(a)). Table 1 shows that the test accuracy of a Resnet-18 model on CIFAR-10 with weights corrupted by PCM noise drops from 95.10% to 10% (random behavior) when fewer than 64 memory cells are used per weight, which does not compare well with our *realistic* baseline for digital storage, where 18 cells per weight would be enough. Luckily, sparsity-driven strategies help to outperform the *realistic* baseline with an 18× improvement in the required amount of storage compared to digital storage. The 5th row of Table 1 shows that we achieve 95.10% accuracy with 3 cells (and 94.44% even with 1 cell) per weight using the sparsity-driven protection; on the other hand, the NN performance is compromised without sparsity-driven protection even using 512 cells per weight with an accuracy of 94.20% (see **No Protection** row) – more than 512× times reduction in the required amount of storage compared to analog storage without our strategies. Figure 5 demonstrates the effectiveness of our approach in preserving NN performance on PCM (Figure 8 for CIFAR-10 in Appendix). We now introduce each sparsity-driven strategies one by one.



(a) Distribution of weights (ResNet-18 trained on CIFAR-10).

(b) Distribution of sensitivity terms (ResNet-18 trained on CIFAR-10).

Fig. 4. Histogram of (a) weights of ResNet-18 trained on CIFAR-10, (b) sensitivity terms ($s_j = \frac{1}{N}\sum_{i=1}^{N}(\frac{\partial \log p_w(y_i|x_i)}{\partial w_j})^2$ from Section 3.1.2) of ResNet-18 trained on CIFAR-10. Since both the weights and sensitivities are sparse, the increase in the average number of cells per weight due to adaptive redundancy and sensitivity-driven protection strategies is negligible.

Table 1. Accuracy of ResNet-18 on CIFAR-10; and ResNet-50, MobileNet-v2, Efficientnet-B2 on ImageNet when Weights are Perturbed by the PCM Cells

		2056 cells	512 cells	64 cells	32 cells	16 cells	8 cells	4 cells	3 cells	2 cells	1 cell	Additional Bits
ResNet-18 CIFAR-10	No Protection	95.1	94.2	27.1	9.0	10.2	10.2	9.9	9.6	9.8	10.3	0
	SP	95.1	95.0	94.2	94.0	92.8	89.5	67.0	41.9	11.8	9.8	1
	AM+AR	95.0	95.0	94.8	94.7	94.4	93.7	93.1	92.7	89.2	58.0	1
	SP+AM	95.1	95.1	95.0	95.0	94.8	94.7	94.6	93.9	93.2	90.6	2
	SP+AM+AR	95.1	95.1	95.0	**95.0**	**95.0**	**95.0**	**95.0**	**95.1**	**94.8**	**94.4**	2
	SP+AM+AR+Sens.	95.1	95.1	95.1	**95.1**	**95.1**	**95.1**	**95.0**	**95.1**	**94.8**	**95.0**	3
ResNet-50 ImageNet	No Protection	67.6	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0
	SP	75.8	4.2	1.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1
	AM+AR	76.1	76.0	70.2	70.0	67.8	65.5	46.1	35.8	10.3	0.1	1
	SP+AM	76.6	75.5	75.0	68.2	65.0	50.2	48.8	25.1	12.2	1.0	2
	SP+AM+AR	**76.6**	**76.6**	**76.6**	**76.6**	**76.6**	**76.4**	**75.9**	**76.0**	**75.8**	**75.5**	2
	SP+AM+AR+Sens.	**76.6**	**76.6**	**76.6**	**76.6**	**76.5**	**76.6**	**76.2**	**75.9**	**76.0**	**75.9**	3
MobileNet-v2 ImageNet	No Protection	34.5	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0
	SP	69.9	2.0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1
	AM+AR	71.4	68.4	65.1	61.0	52.3	36.8	22.6	11.9	0.1	0.1	1
	SP+AM	71.8	71.2	70.6	65.4	50.5	43.8	32.0	16.7	0.1	0.1	2
	SP+AM+AR	**71.8**	**71.8**	**71.8**	**71.8**	**71.8**	**71.4**	**71.0**	**70.5**	**70.2**	**69.5**	2
	SP+AM+AR+Sens.	**71.8**	**71.8**	**71.8**	**71.8**	**71.8**	**71.6**	**71.2**	**71.1**	**71.0**	**70.4**	3
EfficientNet-B2 ImageNet	No Protection	39.8	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0
	SP	72.4	5.2	0.4	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1
	AM+AR	79.6	77.1	72.8	64.7	49.3	37.2	17.5	9.2	0.1	0.1	1
	SP+AM	80.2	78.6	76.5	69.8	57.0	40.8	24.1	15.9	0.1	0.1	2
	SP+AM+AR	**80.2**	**80.2**	**71.8**	**80.2**	**79.9**	**79.2**	**78.6**	**78.0**	**77.4**	**76.4**	2
	SP+AM+AR+Sens.	**80.2**	**80.2**	**80.2**	**80.1**	**80.2**	**80.1**	**79.8**	**79.2**	**78.8**	**77.9**	3

Baseline accuracy (without noise) is 95.1% for ResNet-18 on CIFAR-10, 76.6% for ResNet-50 on ImageNet, 71.8% for MobileNet-v2 on ImageNet, and 80.2% for EfficientNet-B2 on ImageNet. SP: sign protection, AM: adaptive mapping, AR: sparsity-driven adaptive redundancy, Sens.: sensitivity-driven adaptive redundancy. Results are averaged over three runs. Higher is better. We provide the average number of cells, including the ones required to store the additional bits, in Table 2.

Sign Protection: When scaling the weights by α to fit them in range $[-0.65, 0.75]$ of ϕ (Figure 3), small weights are mapped to values very close to zero. This is problematic because a majority of the trained NN weights have small magnitudes, and thus the NN with reconstructed weights will suffer a severe drop in performance due to sign errors (see Figure 4(a)). Therefore, we store the sign and magnitude of the weights separately. The sign can be represented by 1 bit. When we store magnitudes instead of actual weights, we can use an α that is twice as large, reducing the variance of noise from Method #2. The **No Protection** and **SP** rows of Table 1 illustrate the effect

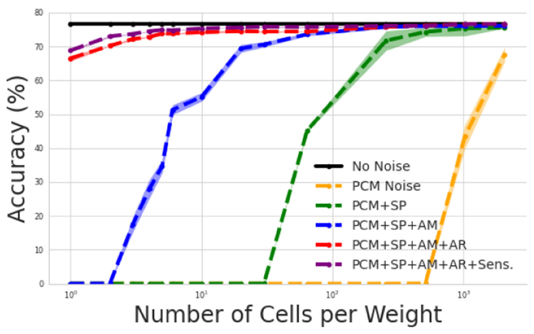


Fig. 5. ResNet-50 on ImageNet. SP: sign protection, AM: adaptive mapping, AR: sparsity-driven adaptive redundancy, AP: adaptive protection (AM+AR), Sens.: sensitivity-driven adaptive redundancy.

of sign protection as the required memory to achieve a test accuracy of 94.1 ± 0.1% on CIFAR-10 is reduced by 16× times (from 512 cells to 32 cells).

Adaptive Mapping: Although protecting the sign bit leads to accuracy gains on CIFAR-10, we still require more than 18 devices to achieve the original accuracy without PCM noise. To further improve the efficiency, we again use the observation that the majority of nonzero weights are quite small (see Figure 4(a)). This implies that using different values of α depending on the magnitude of the weight (larger scale factor α for small weights) can reduce the overall variance of the cell's noise from Method #2. This requires an extra bit to indicate whether a weight is small or large since two different α's are used in encoding and decoding. With this strategy, according to **SP** and **SP+AM** rows of Table 1, we can increase the model's accuracy from 9.8% to 90.6% with an average of 1 cell per weight on CIFAR-10 (and a corresponding increase in accuracy from 0.1% to 51.3% with an average of 16 cells per weight for ImageNet).

Adaptive Redundancy: Finally, we propose to vary the number of PCM cells used for larger and smaller weights. The average number of cells that we aim to minimize is:

$$r_{avg} = \frac{r_{small} \times N_{small} + r_{large} \times N_{large}}{N_{small} + N_{large}}$$

where N_{small} and N_{large} are the number of small and large weights; r_{small} and r_{large} are the number of cells used per weight for small and large weights, respectively. Using more cells for larger weights (which are more critical for NN performance) increases the accuracy while it does not increase the average redundancy too much (Method #1) since most weights are sparse (see Figure 4(a)), e.g., $N_{small} = 11{,}168{,}312$ and $N_{large} = 5{,}650$ in ResNet-18 trained on CIFAR-10. Sign protection and adaptive mapping help protect mostly the small weights while adaptive redundancy protects large weights by reserving more resources for them. As shown in the **SP+AM** and **SP+AM+AR** rows of Table 1, combining these strategies increases the accuracy by 4%/66% on CIFAR-10/ImageNet for the 1 cell per weight case without requiring any additional bits.

3.1.2 *Sensitivity-Driven Protection.*

While we have demonstrated the success of sparsity-driven protection strategies for $g(\cdot)$ even with 1 cell per weight for ResNet-18 on CIFAR-10, the method falls short for more complex datasets. For ImageNet, the accuracy of ResNet-50 with sparsity-driven protection against 1 PCM per weight is 10.6% lower than the original accuracy without PCM noise (76.6%, see Table 1). To close this gap, we first define δ_w as the final perturbation on the weights, i.e., $g(\hat{w}) = w + \delta_w$. Then we approximate the KL divergence via a second-order Taylor expansion:

$$\mathbb{E}_{x\sim p_{\text{data}}(x)}[D_{KL}(p_w(y|x)||p_{w+\delta w}(y|x))] \approx \delta w^T F \delta w + O(||\delta w||^3) \tag{3}$$

where $F := \mathbb{E}_{x,y\sim p_{\text{data}}(x,y)}[\nabla_w \log p_w(y|x) \nabla_w^T \log p_w(y|x)]$ is the Fisher Information Matrix [26, 51]. Dropping the off-diagonal entries in F yields

$$\delta w^T F \delta w \approx \mathbb{E}_{x,y\sim p_{\text{data}}(x,y)} \sum_{j=1}^{d} \left(\delta w_j \cdot \frac{\partial \log p_w(y|x)}{\partial w_j}\right)^2 \tag{4}$$

$$= \sum_{j=1}^{d} \delta w_j^2 \mathbb{E}_{x,y\sim p_{\text{data}}(x,y)} \left(\frac{\partial \log p_w(y|x)}{\partial w_j}\right)^2 \tag{5}$$

where d is the number of weights. Thus, KL divergence between the conditional distribution parameterized by the original (w) and perturbed weights ($g(\hat{w})$) that we aim to minimize is:

$$\mathbb{E}_{x\sim p_{\text{data}}(x)}[D_{KL}(p_w(y|x)||p_{w+\delta w}(y|x))] \approx \frac{1}{N} \sum_{j=1}^{d} \delta w_j^2 \sum_{i=1}^{N} \left(\frac{\partial \log p_w(y_i|x_i)}{\partial w_j}\right)^2 \tag{6}$$

For each weight w_j, we can estimate how much performance degradation δw_j can cause by evaluating the term

$$s_j = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\partial \log p_w(y_i|x_i)}{\partial w_j}\right)^2, \tag{7}$$

which we call "sensitivity". For the storage of sensitive weights w_j with large s_j, we should use more PCM cells as slight perturbations of these weights may lead to significant changes in the output distributions (Method #1). This will not significantly affect the number of cells per weight on average (r_{avg}) because the gradients of a fully trained network (which correlate with the sensitivity) are known to be sparse. Figure 4(b) further demonstrates the sparsity of the sensitivity terms. We combine sensitivity- and sparsity-driven protection to vary the number of memory cells per weight. Table 1 demonstrates that sensitivity-driven protection provides a 2.8% improvement in test accuracy using 1 cell per weight for ResNet-50 trained on ImageNet. Figure 5 illustrates that sparsity-driven and sensitivity-driven protection strategies provide 2056× more efficient storage of NN models by preserving the accuracy against PCM noise. We refer the reader to Figure 8 in the Appendix for similar results on CIFAR-10.

3.2 KL Regularization for Robustness

The following set of techniques for constructing $g(\cdot)$ are designed to correct the errors that the robust coding strategies in the previous section fail to address.

Robust Training: For robust training, we regularize the standard cross-entropy loss with the KL divergence in Equation (1). Specifically, the loss function is as follows:

$$\mathcal{L}(w) = \mathbb{E}_{x,y\sim p_{\text{data}}}[-\log p_w(y|x)] + \lambda \mathbb{E}_x[D_{KL}(p_w(y|x)||p_{g(\hat{w})}(y|x))]. \tag{8}$$

In our experiments, we add a noise (with a carefully adjusted standard deviation) as a perturbation δw (i.e., $g(\hat{w}) = w + \delta w$ is a noisy weight) during robust training, and we observe that the trained network is more robust to PCM noise and also to pruning effects. This is because the final perturbation δw on the weights ($g(\hat{w}) = w + \delta w$) can be thought of as a noise or the effect of pruning on the w. Although our framework does not involve a pruning step (pruning can be performed independently as we show in Section 4.1), we believe that making NN weights more robust to pruning effects is an important additional feature of our strategy. In particular, when we apply pruning, we have: $\delta w_j = 0$ for a non-pruned weight w_j, and $\delta w_j = -w_j$ for a pruned weight w_j. Recall that in magnitude pruning, only the small weights are set to zero, that is δw_j is equal to 0 (for large weights) or $-w_j$ (for small weights). In other words, δw_j is always a small value, just like a noise, therefore this strategy could provide robustness against pruning effects as well. Our experiments on CIFAR-10 and ImageNet verify that robust training has a protective effect against PCM noise where robust coding strategies are not enough.

Robust Distillation: Distillation is a well-explored NN compression technique [33]. The idea is to first train a teacher network to capture a smooth probability distribution on labels, and then train a smaller student network by distilling the output probability information from the teacher. We use distillation to optimize a compressed model to be robust to PCM noise via the noisy student loss:

$$\mathcal{L}_s(w_s) = (1-\lambda)\mathbb{E}_{x,y\sim p_{\text{data}}}[-\log p_{\hat{w}_s}(y|x)] + \lambda\mathbb{E}_x[D_{KL}(p_{w_t}(y|x)||p_{g(\hat{w}_s)}(y|x))] \tag{9}$$

where $\lambda \in [0,1]$ is a scalar, w_t and w_s are teacher and student weights, $g(\hat{w}_s) = g(w_s + \epsilon(w_s)) = w_s + \delta w_s$ with $\epsilon(w_s)$ being the PCM noise and δw_s being the noise injected onto the student network's weights. Although [74] provides an initial exploration into distillation for noisy storage, we leverage experimental data collected from real storage devices to build a realistic model of the PCM noise.

3.3 End-to-End Learning

Finally, we explore a probabilistic, end-to-end learning approach for $g(\cdot)$ that jointly optimizes over the model compression and the known characteristics of the noisy storage device. We assume access to a set of weights $\{W_j\}_{j=1}^{K}$ from K models that have been trained on the same dataset $\mathcal{D}$ – this serves as an empirical distribution over NN weights, as different initializations of NNs typically converge to different local minima. Additionally, we assume that each weight is a sample from a normal distribution $W_{ij} \sim \mathcal{N}(0,1)$ [20, 76].

To learn the representation of the NN weight, we maximize the **mutual information (MI)** between the original network weight W and the compressed weight Z that has been corrupted by the noisy channel. Following [13], our coding scheme can be represented by the following Markov chain[2]:

$$W \rightarrow \hat{Z} \rightarrow Z \rightarrow \hat{W}, \tag{10}$$

where $\hat{Z}$ denotes the compressed weight and $Z = \hat{Z} + \epsilon(Z)$ where $\epsilon(Z)$ denotes the input-dependent PCM noise. Then, we obtain the following lower bound to the true MI:

$$I(W;Z) = H(W) - H(W|Z) = -H(W|Z) + \text{const.} \tag{11}$$

$$\geq \mathbb{E}_{q_\phi(Z|W)}[\log p_\theta(W|Z)] \tag{12}$$

where $q_\phi(Z|W)$ and $p_\theta(W|Z)$ denote approximations to the true posteriors $p(Z|W)$ and $p(W|Z)$, respectively [4]. We train an autoencoder $g_{\phi,\theta}(\cdot)$ with a stochastic encoder and decoder to learn these variational posteriors. The decoder is trained to output a set of reconstructed weights such that its predictions are close to those of the original network. Notably, our approach differs from KD in that we *learn* the weight compression scheme rather than using a fixed student network architecture.

[2]Here, we introduce the new notation $\hat{Z}$ and Z for the compressed and noisy version of the weights because, in the end-to-end learning approach, we have neural encoder and decoder networks to compress ($W \rightarrow \hat{Z}$) and decompress ($Z \rightarrow \hat{W}$) the weights. This notation was not necessary in the previous sections, as we did not have such an encoder-decoder pair.

Table 2. A Sample of Accuracy vs. Average Number of Cells Required to Store: (1) the Continuous Weight Values, and (2) the Additional Bits

	SP+AM+AR	SP+AM+AR	SP+AM+AR	SP+AM+AR+Sens.	SP+AM+AR+Sens.	SP+AM+AR+Sens.
	4 cells	3 cells	2 cells	4.5 cells	3.5 cells	2.5 cells
ResNet-18 on CIFAR-10	95.1	94.8	94.4	95.1	94.8	95.0
ResNet-50 on ImageNet	76.0	75.8	75.5	75.9	76.0	75.9
MobileNet-v2 on ImageNet	70.5	70.2	69.5	71.1	71.0	70.4
EfficientNet-B2 on ImageNet	78.0	77.4	76.4	79.2	78.8	77.9

Numbers taken from Table 1 by computing the average number of cells considering the ones that are required to store the additional bits. Baseline accuracy (without noise) is 95.1% for ResNet-18 on CIFAR-10, 76.6% for ResNet-50 on ImageNet, 71.8% for MobileNet-v2 on ImageNet, and 80.2% for EfficientNet-B2 on ImageNet. SP: sign protection, AM: adaptive mapping, AR: sparsity-driven adaptive redundancy, Sens.: sensitivity-driven adaptive redundancy. Results are averaged over three runs. Higher is better.

4 EXPERIMENTAL RESULTS

We empirically investigate: (1) whether our protection strategies for $g(\cdot)$ help to preserve NN accuracy; and (2) the improvements in storage efficiency when using our approach. All experimental results are averaged over 3–5 runs. For conciseness, we report the average and refer the reader to Appendix B for the complete results.

For all the classification experiments, we consider models trained on three image datasets: MNIST [48], CIFAR-10 [44], and ImageNet [17]. We use the standard train/val/test splits for MNIST and CIFAR-10 datasets and the standard train/val split for the ImageNet dataset. For MNIST, we use two architectures: LeNet [47], and a 3-layer MLP. For CIFAR-10, we use two types of ResNets [32]: ResNet-18 and a slim version of ResNet-20. For ImageNet, we use pretrained ResNet-50 from PyTorch [32], and lightweight models MobileNet-v2 [59] and EfficientNet-B2 [64]. For the regression experiment, we use the standard Neural Radience Fields (NeRF) model [52] on the fern dataset (https://github.com/bmild/nerf). For additional details on architectures and hyperparameters, we refer the reader to Appendix A.

4.1 Sparsity and Sensitivity Driven Protection

We show the effect of sparsity- and sensitivity-driven protection in Figure 5 on ResNet-50 for ImageNet (Figure 8 for CIFAR-10 in Appendix). The exact numerical results are given in Table 1.

In CIFAR-10 experiments, the number of small weights was N_{small} = 11M and the number of large weights was N_{large} = 5.6K and number of cells per weight on average for adaptive redundancy is not more than 0.02 above the listed numbers in the table. Similarly, in ImageNet experiments, N_{small} = 25.4M, N_{large} = 15K and the difference between the number of cells per weight on average and the listed number is always smaller than 0.06.

In Table 1, we provide detailed experimental results on the effect of each sparsity- and sensitivity-driven strategy with ResNet-10 on CIFAR-10; and ResNet-50, MobileNet-v2, and EfficientNet-B2 on ImageNet. We provide the required number of cells per weight to store the continues value of the weight (3rd-12th columns) and the additional bits to store for each strategy (last column) separately. In Table 2, we report some of the results from Table 1 again, this time by providing the number of cells per parameter required to store (1) the continuous weight value, and (2) the additional bits. Recall that we can store 2 bits or 1.8 bits digitally in one PCM cell with the *ideal* or *realistic* baselines, respectively. It is seen from the two tables that sparsity driven protection strategies are enough for ResNet-18 on CIFAR-10 to preserve the classification accuracy (95.10%) with 4 cells per weight. Moreover the accuracy with 2.5 cells per weight. is only 0.1% less than the baseline accuracy (95.10%). Similar observation can be made for the ImageNet results as well. Table 1 also shows that sign protection is a critical step: for instance, the **AM+AR** and **SP+AM+AR** rows indicate that sign protection can increase the accuracy from 58% **AM+AR** to 94.4% **SP+AM+AR** on CIFAR-10.[3]

We also combine sparsity driven protection with pruning. Results on 90% pruned ResNet-18 (on CIFAR-10) are given in Table 15 in Appendix B.2. We follow the standard pruning procedure: first train the model as usual, then prune 90% of the weights, and then retrain the remaining non-pruned weights by keeping the pruned weights frozen at value zero. In our experiment, we retrain the pruned model for 20 epochs. Then for the storage of this pruned model, we assume that the pruning mask could be stored using the **compressed sparse row (CSR)** or **compressed sparse column (CSC)** format and Huffman coding, as detailed in [28]. This would require at most 0.5 bits per weight. Then we need to store the continuous weight values of the *non-pruned weights*. Note that this is the standard technique for storing a pruned network: (1) first store the pruning mask, (2) then store the values of the non-pruned weights. In Table 15, the number of cells we report are computed as follows:

$$\text{avg. \# of cells per weight} = \frac{\text{\# of pruned weights}}{\text{total \# of weights}} \times (\text{avg. \# of cells per non-pruned weight}) + 0.25,$$

translating the number of cells per non-pruned weight to number of cells per (pruned or non-pruned) weight and also considering additional 0.25 cells per parameter to store the pruning mask. Note that the accuracy of the pruned model without noise is 94.8% and we get 94.2% test accuracy with PCM noise using only 1.35 cells per weight. Compared to the *realistic* digital baseline (no compression and each 32-bit weight is digitally stored on 18 PCM cells), we provide $\frac{18}{1.35} = 13.3\times$ more efficient storage with analog storage combined with our strategies and 90% pruning.

4.2 Robust Training and Distillation

We compare robust training with naive training (training with no noise) in Table 3 on ResNet-18 (on CIFAR-10) and on ResNet-50 (on ImageNet). In our experiments, we use $\lambda = 0.5$ as the coefficient of the KL term in the loss. We experimented with different combinations of the sparsity and sensitivity driven strategies, and in all cases, robust training provides better robustness against PCM noise than the naive training. Robust training provides robustness against pruning as well. When ResNet-18 trained without noise is pruned with 90% sparsity, the accuracy drops from 95.1% to **90.2**%. However, the same model trained with $N(0, 0.006)$ gives **95.0**% accuracy after 90% pruning.

Table 4 shows the distillation results on ResNet-20 distilled from ResNet-18 (on CIFAR-10) with PCM noise applied at test time. We compare three networks: (1) a student ResNet-20 distilled without noise injection, (2) a student ResNet-20 distilled with Gaussian noise injection, and (3) a teacher ResNet-20 (a baseline) trained without noise. As shown in Table 4, student ResNet-20 distilled with noise injection outperforms both teacher ResNet-20 and student ResNet-20 distilled without noise when weights are perturbed by PCM at test time. (see Appendix B.4 for additional results.)

4.3 Analog-Digital Comparison

We also consider quantization as a way to improve the efficiency of digital storage. Table 5 shows a comparison between analog storage improved with our strategies and digital storage improved

[3]We would like to note that it is possible to improve the performance of the noisy NNs further by retraining them after the weights are read from the PCM cells. In fact, our experiments suggest that for the models stored with SP+AM+AR and SP+Am+AR+Sens. strategies using 1 cell per weight, retraining recovers the original accuracy in $1-3$ epochs. However, we believe it is not realistic to assume that the stored NNs can be retrained further since we are particularly interested in resource-constrained edge devices.

Table 3. Robust Training Vs. Naive Training for ResNet-18 on CIFAR-10 and ResNet-50 on ImageNet

		Naive Train (no noise)	Robust Train (with $\sigma = 0.01$)	Robust Train (with $\sigma = 0.006$)	Average Number of Cells
CIFAR-10	No Noise	95.10	95.50	**95.60**	-
	PCM (No Protection)	9.70	8.30	9.90	1
	PCM+SP	9.70	10.63	10.33	1.5
	PCM+AP	27.69	**86.20**	81.83	1.5
	PCM+SP+AP	90.60	94.73	**94.80**	2
	PCM+AP+Sens	36.21	**86.73**	78.93	2
	PCM+SP+AP+Sens	94.95	**95.03**	**95.03**	2.5
ImageNet	No Noise	76.60	76.60	76.60	-
	PCM (No Protection)	0.1	0.1	0.1	1
	PCM+SP	0.1	**0.4**	0.2	1.5
	PCM+AP	0.1	**0.002**	0.1	1.5
	PCM+SP+AP	75.50	77.10	**77.80**	2
	PCM+AP+Sens	0.3	**0.6**	0.5	2
	PCM+SP+AP+Sens	75.90	76.20	**76.50**	2.5

SP: sign protection, AP: adaptive protection (adaptive mapping+sparsity-driven adaptive redundancy), Sens.: sensitivity-driven adaptive redundancy. The average number of cells per weight to store the continuous weight value is 1 in all experiments. The reported average number of cells in the rightmost column includes (1) the number of cells required to store the continuous weight value, and (2) the number of cells required to store the additional bits.

Table 4. Accuracy of ResNet-20 Distilled from ResNet-18 on CIFAR-10

	Avg. # of Cells for Cont. Weights	Teacher ResNet-18	Teacher ResNet-20	Student ResNet-20	Noisy Student ResNet-20	Avg. # of Cells in Total
No Noise	-	95.70	92.50	92.90	**93.00**	-
PCM+AP	3	16.23	48.38	73.38	**81.75**	3.5
PCM+SP+AP	1	93.35	86.30	88.58	**90.65**	2
	3	94.78	89.73	89.98	**91.33**	4
PCM+AP+Sens.	1	9.60	29.68	38.18	**69.49**	2
PCM+SP+AP+Sens.	1	93.36	88.40	88.96	**91.10**	2.5
	3	94.90	89.92	90.44	**91.78**	4.5

The average number of cells per weight to store the continuous weight value is given in the leftmost column. The reported average number of cells in the rightmost column includes (1) the number of cells required to store the continuous weight value, and (2) the number of cells required to store the additional bits.

with quantization techniques. We consider both the *ideal* (2 bits per cell) and *realistic* baselines (1.8 bits per cell). For digital storage, we apply quantization using different techniques from the literature [3, 41]. We find the number of cells to store one quantized weight in both *ideal* and *realistic* cases. Then we adjust the number of cells used to store one weight in analog PCMs to be the same as the digital PCMs for both baselines, by adapting the redundancy for large and less sensitive weights. For instance, when [3] performs 4-bit quantization, each parameter is represented by 4 bits. In the *ideal* baseline, this would require 2 cells per parameter. We adjust the parameters in our robust strategies so that the number of cells required to store one parameter is 2 and report the result (76.02) in the "PCM (analog) + our robust strategies" row under "*Ideal*" column. We repeat the same procedure for the *realistic* baseline which requires 2.22 cells per parameter to store 4 bits. We report the result (76.08) in the "PCM (analog) + our robust strategies" row under "*Realistic*" column. As shown in Table 5, noisy analog storage improved by our strategies outperforms digital storage of quantized weights. We do not compare against more aggressive quantization techniques [2, 14, 23, 43, 54, 66] that can achieve higher efficiency in digital storage since they incur a huge complexity with multiple retraining stages.

Table 5. Digital vs. Analog Storage for ResNet-50 on ImageNet

		Ideal	*Realistic*
~4 cells per parameter	PCM (digital) + 8-bit quantization with [41]	68.30	68.30
	PCM (digital) + 8-bit quantization with ACIQ [3]	73.60	73.60
	PCM (analog) + our robust strategies	**76.02**	**76.08**
~2 cells per parameter	PCM (digital) + 4-bit quantization with [41]	72.50	72.50
	PCM (digital) + 4-bit quantization with ACIQ [3]	73.80	73.80
	PCM (analog) + our robust strategies	**75.50**	**75.62**

For digital storage, number of cells is reduced via 4-bit and 8-bit quantization techniques [3, 41]. For analog storage, it is reduced via our strategies. For fair comparison, the number of PCM cells per weight in analog storage is adjusted to be the same as the digital storage (in both *ideal* and *realistic* baselines). For instance, when [3] performs 4-bit quantization, each parameter is represented by 4 bits. In the *ideal* baseline, this would require 2 cells per parameter. We adjust the parameters in our robust strategies so that the number of cells required to store one parameter is 2 and report the result (76.02) in the "PCM (analog) + our robust strategies" row under "*Ideal*" column. We repeat the same procedure for the *realistic* baseline which requires 2.22 cells per parameter to store 4 bits. We report the result (76.08) in the "PCM (analog) + our robust strategies" row under "*Realistic*" column.

Table 6. NeRF [52] Model on Fern Dataset (https://github.com/bmild/nerf)

	PSNR (dB)
No protection (32 cells)	5.66
SP+AM+AR (4 cells)	23.08
SP+AM+AR (3 cells)	20.41
SP+AM+AR (2 cells)	18.20
SP+AM+AR+Sens. (4.5 cells)	24.75
SP+AM+AR+Sens. (3.5 cells)	22.98
SP+AM+AR+Sens. (2.5 cells)	20.65

A sample of PSNR vs. average number of cells required to store (1) the continuous weight values, and (2) the additional bits. Baseline PSNR (without noise) is 24.73 dB. SP: sign protection, AM: adaptive mapping, AR: sparsity-driven adaptive redundancy, Sens.: sensitivity-driven adaptive redundancy. Higher is better.

4.4 Regression Models

We test our strategies on a regression setting as well. For the regression task, we consider the Neural Radiance Fields (NeRF) framework [52] that contains a neural network to model the relation between (1) the 3D coordinates of a point in a given 3D scene and the view direction; and (2) the view-dependent continuous color[4] and volume density at that 3D point. More precisely, a NeRF model takes 3D positions (x, y, z) of a point in 3D space and a particular direction to view the 3D scene (θ, ϕ); and outputs a view-dependent RGB color and a volume density at that point. We note that this is a nontrivial task that has been attracting significant interest from the computer graphics and vision communities since the first paper in 2020 [52]. We report the PSNR of the NeRF model on fern dataset (https://github.com/bmild/nerf) that was generated by the authors of [52] under the effect of PCM noise in Table 6. Though the PSNR is affected severely by the PCM noise when there is no protection, our robust strategies help to preserve the PSNR. In particular, the PSNR is 24.75 dB with an average number of 4.5 cells, which is even higher than the baseline

[4]Color is then discretized during evaluation.

Table 7. MLP-based Autoencoder Architecture for Synthetic Experiments, Trained on 50 Logistic Regression Classifiers Per Task

Name	Component
(Encoder) Input Layer	Linear 3 $\rightarrow$ 1, ReLU
(Encoder) Hidden Layer	Linear 1 $\rightarrow$ 1
(Decoder) Hidden Layer	Linear 1 $\rightarrow$ 1, ReLU
(Decoder) Output Layer	Linear 1 $\rightarrow$ 3

(without noise) PSNR 24.73 dB. Additionally, we note that we would normally expect to see a more severe effect on PSNR by the PCM noise. This is because the exact values of the model outputs matter more in a regression setting than in a discriminative one, since the classifier's accuracy can remain unchanged even with small perturbations to the logits. However, as can be seen from Table 6, PSNR is still within an acceptable range once we apply the robust strategies. We believe this is due to the discretization at the evaluation phase, which may cancel out some noise on the output.

As a final note, the storage density for the NeRF model or other 3D regression models could be improved further through model compression techniques such as [7, 35, 36].

4.5 End-to-End Learning

Finally, we explore the effectiveness of the end-to-end learning scheme in which we train an autoencoder on a *set of model weights*. We generate two sets of 2-D Gaussian mixtures: an "easy" task in which we first sample a two-dimensional mean vector $\mu_1 \in \mathbb{R}^2$ where $\mu_{1i} \sim \mathcal{U}[-1, 0)$ and $\mu_2 \in \mathbb{R}^2$ where $\mu_{2i} \sim \mathcal{U}[0, 1)$ for $i = \{1, 2\}$. After sampling these means, we draw a set of 50K points for the two mixture components: $x_1 \sim \mathcal{N}(\mu_1, I)$ and $x_2 \sim \mathcal{N}(\mu_2, I)$. This ensures that the two mixture components are well-separated. For the "hard" task, we sample overlapping means: both components of μ_1 and μ_2 are drawn from $\mathcal{U}[-1, +1]$ before sampling from their respective mixture components. We generate 50 datasets per task, where each dataset has 50K data points.

We train 50 different logistic regression models on each of the datasets for both the "easy" and "hard" tasks (each model has three parameters). After training the logistic regression models, we use an autoencoder with MLP encoder and MLP decoder architectures and ReLU nonlinearities, as shown in Table 7. The autoencoder for both the "easy" and "hard" tasks is trained for 10 epochs with a batch size of 100 using the Adam optimizer with learning rate = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and early stopping on a held-out validation set.

We test the autoencoder on both Gaussian and PCM noise: that is, we corrupt the 1-dimensional encoder output (z-representation of the classifier weights) with the appropriate perturbation before passing in the encoded representation into the decoder. Figure 6 provides an illustration of the autoencoding process for the PCM array, which is analogous to the Gaussian noise setup. For the easy task with Gaussian noise, the autoencoder achieves an accuracy of 95.2%; for PCM noise, it achieves 91.8% accuracy. For the hard task with Gaussian noise, the autoencoder achieves an average accuracy of 78.8% across all classifiers; for PCM channel noise, it achieves 78.6% on average across all 50 classifiers.

Interestingly, we find that the 1-D classifier representations in the autoencoder's latent space is semantically meaningful. In Figure 7, we qualitatively analyze the learned representations of the logistic regression classifier weights. In Figure 7(a), we plot all 50 datasets of the Gaussian mixtures ("easy task") as well as the true decision boundaries for each of the 50 logistic regression models, each boundary colored by the magnitude of its z-representation as learned by the autoencoder. We plot the same for the "hard task" in Figure 7(b). For the easy task, we note that

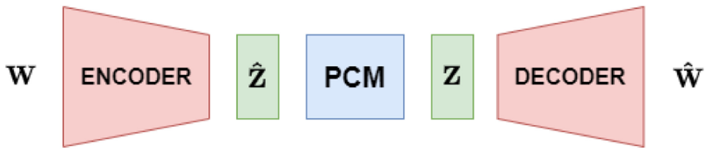
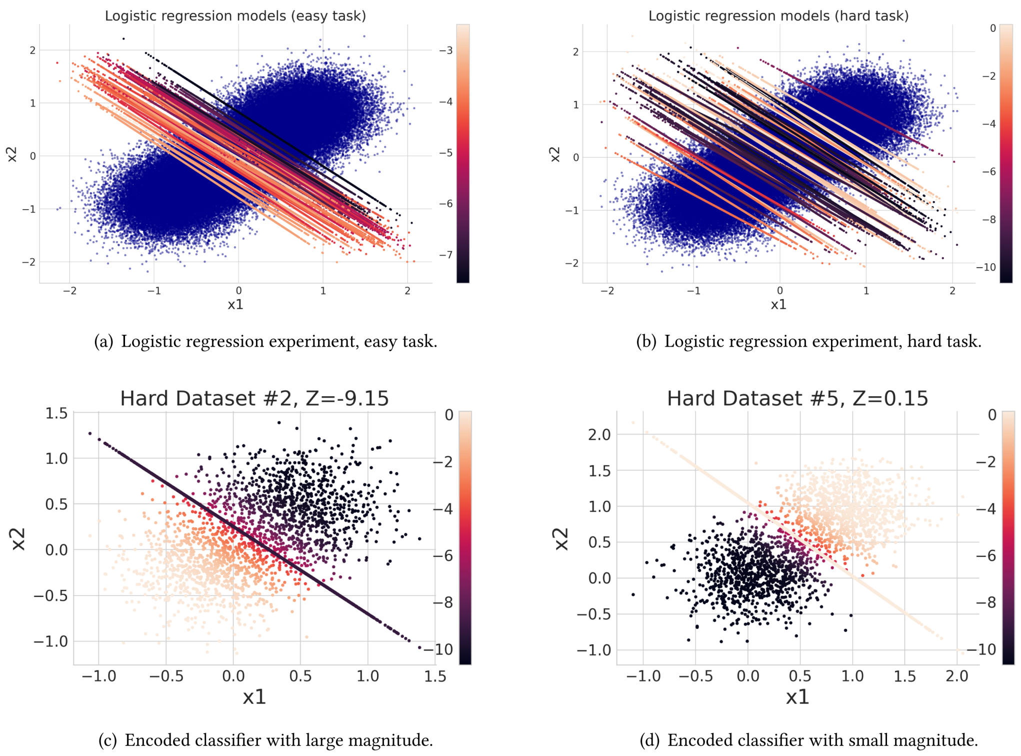


Fig. 6. Illustration of the autoencoder used for the logistic regression experiments. The input weight W is mapped to a compressed representation $\hat{Z}$ by an encoder module, which is then perturbed by the PCM (or analogously, Gaussian) noise channel to become a perturbed representation Z. This perturbed representation is then passed to the decoder, which produces a reconstructed weight $\hat{W}$.



(a) Logistic regression experiment, easy task.

(b) Logistic regression experiment, hard task.

(c) Encoded classifier with large magnitude.

(d) Encoded classifier with small magnitude.

Fig. 7. **(a-b)** Qualitative visualizations of the learned representations in the logistic regression experiment. (a) shows all 50 datasets with the true decision boundaries colored by the magnitude of their z-representations for the "easy task", while (b) shows the analogous plot for the harder task with overlapping means. **(c-d)** Qualitative visualizations of the learned representations for the hard logistic regression task. We find in (c) that classifiers with large magnitudes in z-space have the positive labels to the left of the decision boundary, while (d) those with small magnitudes have the positive labels to the right of the decision boundary.

the classifiers are encoded by their relative location in input-space: those that are in the lower left corner of the scatter plot have smaller magnitudes than those on the upper right. For the hard task, however, the z-representations appear to be fairly random – at a first glance, there does not appear to be a particular correlation between the magnitudes of the z-encodings and the original classifier weights.

We further explore this phenomenon for the hard task in Figure 7(c-d). For two particular datasets (though the trend holds across all 50 datasets), we color the original data points by their mixture component as well as the true decision boundary by the magnitude of its z-encoding. As shown in Figure 7(c) and (d), we find that the autoencoder has learned to map all classifiers with the positive class to the left of the decision boundary to z-representations with large magnitude; conversely, those with the positive class to the right of the decision boundary are encoded to z-representations with smaller magnitudes. Through this preliminary investigation, we demonstrate that the end-to-end approach for learning both the compression scheme while taking the physical constraints of the storage device into account shows promise.

5 RELATED WORK

In this section, we briefly summarize the related work in model compression and analog computing in NNs.

Model Compression. Compression of NN parameters has a rich history starting from the early works of [15, 30], with techniques such as pruning [5, 25, 28, 29, 37, 38, 40, 45, 49, 56, 61, 63], quantization, [2, 14, 23, 43, 70], and KD [33, 57, 69] among others [18]. [55] explores KD as a way to encourage robustness to adversarial perturbations in input space, while we specifically desire robustness to weight perturbations. Although probabilistic approaches to model compression have been explored in [31, 50, 58], we additionally consider the physical constraints of the storage device used for memory as part of our learning scheme. Our end-to-end approach is most similar to [54], in that they also learn a decoder to map NN weights into a latent space prior to quantization. However, our method is different in that our autoencoder also learns the appropriate compression scheme (with an encoder), we inject noise into our compressed weights rather than quantizing, and we do not require training an NN from scratch.

Analog Computing in NNs. A complementary line of work utilizes analog components in NN training and/or inference [6, 8, 10, 19, 42, 60, 75]. The common theme is performing computation in the analog domain to reduce the overall computation power, but this procedure may be noisy or inflexible. In contrast, our work focuses on *storing* NN models in analog cells – once the parameters are read from memory, the actual computation happens in the digital domain.

6 DISCUSSION AND CONCLUSION

In this work, we formalized the problem of jointly optimizing model compression and memory resource allocation on noisy analog storage devices. We introduced novel coding techniques for preserving NN accuracy even in the presence of weight perturbations, and demonstrated their effectiveness on models trained on MNIST, CIFAR-10, and ImageNet. Additionally, we explored different training strategies that can be coupled with existing compression techniques such as distillation, and provided an initial foray into a fully end-to-end learning scheme for the joint problem.

Limitations. First, the actual deployment of our approach may require some level of quantization in practice – directly writing analog values on a physical storage device requires complex read/write circuitry, which may not be feasible on current systems. For future work, we aim to investigate this bottleneck on physical hardware devices. Second, our end-to-end learned compression scheme assumes that all models share identical structures and are trained on the same dataset. Extending our framework to handle models trained on various datasets/tasks is an exciting research direction.

Broader Impacts. cAlthough our work aims to reduce power and memory consumption through more efficient NN model compression, it is still susceptible to propagating existing biases in the original trained network. While the research community has mainly evaluated the success of NN compression methods by only considering the compression ratio-accuracy trade-off, a recent study has shown that *existing compression methods may disproportionately impact different subgroups of the data* [34]. Unfortunately, this implies that we will not be able to detect or prevent potential amplification of existing biases once the network is deployed. This speaks to the fundamental importance in the careful curation of datasets and selection of training objectives to mitigate model bias.

APPENDICES

A ADDITIONAL EXPERIMENTAL DETAILS

We conducted our experiments on NVIDIA Titan XP GPUs on an internal cluster server. We used 2 GPUs for ImageNet experiments and 1 GPU for the rest of the experiments. In the following subsection, we provide additional details on the models, model architectures, and hyperparameters used in our experiments.

A.1 MNIST

The architectures for LeNet and the MLP are shown in Tables 8 and 9, respectively:

Table 8. LeNet Architecture for MNIST Experiments

Name	**Component**
conv1	[5×5 conv, 20 filters, stride 1], ReLU, 2×2 max pool
conv2	[5×5 conv, 50 filters, stride 1], ReLU, 2×2 max pool
Linear	Linear 800 $\rightarrow$ 500, ReLU
Output Layer	Linear 500 $\rightarrow$ 10

Table 9. MLP Architecture for MNIST Experiments

Name	**Component**
Input Layer	Linear 784 $\rightarrow$ 100, ReLU
Hidden Layer	Linear 100 $\rightarrow$ 100, ReLU
Output Layer	Linear 100 $\rightarrow$ 10

For both the LeNet and MLP classifiers, we use a batch size of 100 and train for 100 epochs, early stopping at the best accuracy on the validation set. We use the Adam optimizer with learning rate $= 0.001$, and $\beta_1 = 0.9, \beta_2 = 0.999$ with weight decay $= 5e^{-4}$. For knowledge distillation, we use a temperature parameter of $T = 1.5$ and equally weight the contributions of the student network's cross entropy loss and the distillation loss ($\lambda = 0.5$).

A.2 CIFAR-10

We provide the architectural details for the ResNet-18 and the slim ResNet-20 used in our experiments in Tables 10 and 11 below:

Table 10. ResNet-18 Architecture for CIFAR-10 Experiments

Name	**Component**
conv1	3×3 conv, 64 filters. stride 1, BatchNorm
Residual Block 1	$\begin{bmatrix} 3 \times 3 \text{ conv, 64 filters} \\ 3 \times 3 \text{ conv, 64 filters} \end{bmatrix} \times 2$
Residual Block 2	$\begin{bmatrix} 3 \times 3 \text{ conv, 128 filters} \\ 3 \times 3 \text{ conv, 128 filters} \end{bmatrix} \times 2$
Residual Block 3	$\begin{bmatrix} 3 \times 3 \text{ conv, 256 filters} \\ 3 \times 3 \text{ conv, 256 filters} \end{bmatrix} \times 2$
Residual Block 4	$\begin{bmatrix} 3 \times 3 \text{ conv, 512 filters} \\ 3 \times 3 \text{ conv, 512 filters} \end{bmatrix} \times 2$
Output Layer	4×4 average pool stride 1, fully-connected, softmax

Table 11. Slim ResNet-20 Architecture for CIFAR-10 Experiments

Name	**Component**
conv1	3×3 conv, 16 filters. stride 1, BatchNorm
Residual Block 1	$\begin{bmatrix} 3 \times 3 \text{ conv, 16 filters} \\ 3 \times 3 \text{ conv, 16 filters} \end{bmatrix} \times 2$
Residual Block 2	$\begin{bmatrix} 3 \times 3 \text{ conv, 32 filters} \\ 3 \times 3 \text{ conv, 32 filters} \end{bmatrix} \times 2$
Residual Block 3	$\begin{bmatrix} 3 \times 3 \text{ conv, 64 filters} \\ 3 \times 3 \text{ conv, 64 filters} \end{bmatrix} \times 2$
Output Layer	7×7 average pool stride 1, fully-connected, softmax

For both ResNet-18 and slim ResNet-20, we use a batch size of 128 and train for 350 epochs, early stopping at the best accuracy on the validation set. We use SGD with learning rate = 0.1, and momentum = 0.9 and weight decay = $5e^{-4}$.

A.3 ImageNet

We provide the architectural details for the ResNet-50 used in our experiments in Table 12.

Table 12. ResNet-50 Architecture for ImageNet Experiments

Name	Component
conv1	3×3 conv, 64 filters. stride 1, BatchNorm
Residual Block 1	$\begin{bmatrix} 1 \times 1 \text{ conv, 64 filters} \\ 3 \times 3 \text{ conv, 64 filters} \\ 1 \times 1 \text{ conv, 256 filters} \end{bmatrix} \times 3$
Residual Block 2	$\begin{bmatrix} 1 \times 1 \text{ conv, 128 filters} \\ 3 \times 3 \text{ conv, 128 filters} \\ 1 \times 1 \text{ conv, 512 filters} \end{bmatrix} \times 4$
Residual Block 3	$\begin{bmatrix} 1 \times 1 \text{ conv, 256 filters} \\ 3 \times 3 \text{ conv, 256 filters} \\ 1 \times 1 \text{ conv, 1024 filters} \end{bmatrix} \times 6$
Residual Block 4	$\begin{bmatrix} 1 \times 1 \text{ conv, 512 filters} \\ 3 \times 3 \text{ conv, 512 filters} \\ 1 \times 1 \text{ conv, 2048 filters} \end{bmatrix} \times 3$
Output Layer	4×4 average pool stride 1, fully-connected, softmax

We use the pretrained ResNet-50 from PyTorch (https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py), with a batch size of 64. For the robust training experiments, we retrain the model for 20 epochs with early stopping at best accuracy. We use SGD with learning rate = 0.001, and momentum = 0.9 and weight decay = $5e^{-4}$.

Table 13. Accuracy of ResNet-18 on CIFAR-10 when Weights are Perturbed by the PCM Cells

	512 cells	64 cells	32 cells	16 cells	8 cells	4 cells	3 cells	2 cells	1 cell	Additional Bits
No Protection	94.2(± 0.1)	27.1 (± 3.4)	9.0 (± 4.1)	10.2 (± 0.5)	10.2 (± 0.5)	9.9 (± 0.1)	9.6 (± 0.5)	9.8 (± 0.7)	10.3 (± 0.4)	0
SP	95.0 (± 0.0)	94.2 (± 0.1)	94.00 (± 0.2)	92.80 (± 0.3)	89.50 (± 0.6)	67.00 (± 0.2)	41.90 (± 0.5)	11.80 (± 5.1)	9.80 (± 1.5)	1
AM+AR	95.0 (± 0.0)	94.8 (± 0.1)	94.70 (± 0.1)	94.40 (± 0.1)	93.70 (± 0.2)	93.10 (± 0.2)	92.70 (± 0.2)	89.20 (± 2.6)	58.00 (± 8.2)	1
SP+AM	95.1 (± 0.0)	95.0 (± 0.0)	95.00 (± 0.1)	94.80 (± 0.1)	94.70 (± 0.1)	94.60 (± 0.1)	93.90 (± 0.2)	93.20 (± 0.1)	90.60 (± 0.3)	2
SP+AM+AR	95.1 (± 0.0)	95.0 (± 0.0)	**95.00** (± 0.1)	**95.00** (± 0.1)	**95.00** (± 0.1)	**95.00** (± 0.1)	**95.10** (± 0.1)	**94.80** (± 0.1)	**94.44** (± 0.2)	2
SP+AM+AR+Sens.	95.1 (± 0.0)	95.1 (± 0.0)	**95.10** (± 0.1)	**95.07** (± 0.1)	**95.11** (± 0.1)	**95.03** (± 0.2)	**95.14** (± 0.1)	**94.80** (± 0.1)	**94.95** (± 0.3)	3

Baseline accuracy (when there is no noise) is 95.10%. SP: sign protection, AM: adaptive mapping, AR: sparsity-driven adaptive redundancy, Sens.: sensitivity-driven adaptive redundancy. Reported results are averaged over three experimental runs.

B ADDITIONAL EXPERIMENTAL RESULTS

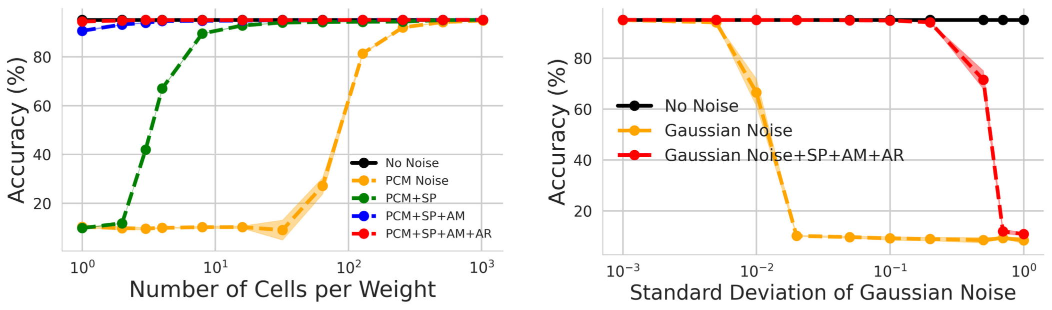
B.1 Sparsity- and Sensitivity-Driven Protection

We give the full results of sparsity-driven and sensitivity-driven protection experiments on CIFAR-10 and ImageNet with confidence intervals in Tables 13 and 14. In Figure 8, we present experimental results with ResNet-18 on CIFAR-10. As can be seen from Figure 8(a), our strategies, namely sign protection, adaptive mapping, and adaptive redundancy, reduce the number of PCM cells per weight required to preserve accuracy from 1,024 to 1. We also test our strategies against Gaussian noise. In the Gaussian experiments, we consider hypothetical storage devices with white Gaussian noise where the standard deviation of the overall noise on the output ($\hat{\epsilon}(r, \alpha)$ "not input-dependent") can be reduced by using the channel multiple times (Method #1) and by using the channel in the its power limit (Method #2). Figure 8(b) shows that our strategies increase the standard deviation threshold, where the NN performance sharply drops, from 0.005 to 0.2, i.e.,

Table 14. Accuracy of ResNet-50 on ImageNet when Weights are Perturbed by the PCM Cells

	512 cells	64 cells	32 cells	16 cells	8 cells	4 cells	3 cells	2 cells	1 cell	Additional Bits
No Protection	0.1(± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0
SP	4.4(± 0.0)	0.9 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	1
AM+AR	75.0 (± 0.5)	66.7 (± 0.8)	66.4 (± 2.5)	63.2 (± 2.5)	57.5 (± 2.2)	34.1 (± 3.4)	22.4 (± 3.0)	3.7 (± 0.0)	0.1 (± 0.0)	2
SP+AM	70.6 (± 0.7)	69.4 (± 1.1)	55.1 (± 3.2)	51.3 (± 2.4)	34.6 (± 1.8)	28.0 (± 5.6)	17.6 (± 4.1)	4.6 (± 0.0)	0.2 (± 0.0)	2
SP+AM+AR	75.4 (± 0.2)	74.5 (± 0.1)	74.2 (± 0.4)	73.8 (± 0.0)	73.8 (± 0.2)	72.8 (± 0.0)	72.2 (± 0.4)	70.2 (± 0.0)	66.0 (± 0.8)	1
SP+AM+AR+Sens.	**75.8** (± 0.0)	**75.6** (± 0.0)	**75.3** (± 0.0)	**74.8** (± 0.1)	**74.9** (± 0.1)	**74.5** (± 0.1)	**73.7** (± 0.0)	**73.0** (± 0.2)	**68.8** (± 0.3)	3

Baseline accuracy (when there is no noise) is 76.6%. SP: sign protection, AM: adaptive mapping, AR: sparsity-driven adaptive redundancy, Sens.: sensitivity-driven adaptive redundancy. Reported results are averaged over three experimental runs.



(a) Perturbation by PCM cells. (b) Perturbation by Gaussian noise.

Fig. 8. Accuracy of ResNet-18 on CIFAR-10 when weights are perturbed by (a) PCM cells, (b) Gaussian noise. SP: sign protection, AM: adaptive mapping, AR: sparsity-driven adaptive redundancy. Experiments are conducted three times.

robustness increases by 40 times. We note that we present results of Gaussian noise experiments to be comparable to future work although this scenario is not realistic.

B.2 Robust Pruning

We give the results of robust pruning experiment in Table 15. We apply one-shot pruning followed by 20 epochs of retraining. When sparsity- and sensitivity-driven strategies are applied, 2.05 cells per weight are enough to preserve the original accuracy of the pruned model (94.8%). Moreover, if we use only 1.35 cells per weight on average, the accuracy drop is only 0.1%, which is insignificant considering the fact that pruning reduces the accuracy by 0.3% (from 95.1% to 94.8%). We note that this performance degradation due to pruning can be eliminated with robust training, explained in Section 3.2. With this strategy, we can reduce the number of cells required to store each weight from 16 (in digital storage) to 1.35 with analog storage combined with our robust strategies and pruning. This corresponds to an 11.85 times more efficient storage.

Table 15. Accuracy of 90% Pruned ResNet-18 on CIFAR-10 when Weights are Perturbed by the PCM Cells vs the Average Number of Cells Required to Store (1) the Continues Weight Values of the Non-pruned Parameters, and (2) the Pruning Mask

	2.05 cells	1.55 cells	1.95 cells	1.45 cells	1.85 cells	1.35 cells
SP+AM+AR	-	94.3	-	94.5	-	94.2
SP+AM+AR+Sens.	94.8	-	94.6	-	94.7	-

Baseline accuracy after the pruning (when there is no noise) is 94.8%. SP: sign protection, AM: adaptive mapping, AR: sparsity-driven adaptive redundancy, Sens.: sensitivity-driven adaptive redundancy.

Table 16. Accuracy of ResNet-18 on CIFAR-10 when Weights are Perturbed by the PCM Cells

	Naive Training (no noise)	Robust Training (with N(0, 0.01))	Robust Training (with N(0, 0.006))	Additional Bits
No Noise	95.10	95.50	**95.60**	0
PCM Noise (No Protection)	9.70(± 1.0)	8.30 (± 0.4)	9.90 (± 1.1)	1
PCM Noise+SP	9.70(± 1.5)	10.63 (± 0.2)	10.33 (± 0.2)	2
PCM Noise+SP+AP	90.60(± 0.2)	94.73 (± 0.1)	**94.80** (± 0.2)	2
PCM Noise+AP	27.69(± 8.2)	**86.20** (± 0.4)	81.83 (± 4.2)	1
PCM Noise+AP+Sens	36.21(± 3.1)	**86.73** (± 2.2)	78.93 (± 4.4)	2
PCM Noise+SP+AP+Sens	94.95(± 0.3)	**95.03** (± 0.1)	**95.03** (± 0.1)	3

Baseline accuracy (when there is no noise at train or test time) is 95.10%. SP: sign protection, AP: adaptive protection (adaptive mapping+sparsity-driven adaptive redundancy), Sens.: sensitivity-driven adaptive redundancy. Experiments are conducted three times.

Table 17. Accuracy of ResNet-50 on ImageNet when Weights are Perturbed by the PCM Cells

	Naive Training (no noise)	Robust Training (with N(0, 0.01))	Robust Training (with N(0, 0.006))	Additional Bits
No Noise	76.60	76.60	76.60	0
PCM Noise (No Protection)	0.1(± 0.0)	0.1 (± 0.0)	0.1 (± 0.0)	0
PCM Noise+SP	0.1(± 0.0)	**0.4** (± 0.0)	0.2 (± 0.0)	1
PCM Noise+AP	0.1(± 0.0)	**0.2** (± 0.0)	0.1 (± 0.0)	1
PCM Noise+SP+AP	66.00(± 0.8)	69.00 (± 0.3)	**69.20** (± 0.2)	2
PCM Noise+AP+Sens	0.3(± 0.0)	**0.6** (± 0.0)	0.5 (± 0.0)	2
PCM Noise+SP+AP+Sens	68.80 (± 0.3)	**70.20** (± 0.0)	**70.40** (± 0.1)	3

Baseline accuracy (when there is no noise at train or test time) is 74.4%. SP: sign protection, AP: adaptive protection (adaptive mapping+sparsity-driven adaptive redundancy), Sens.: sensitivity-driven adaptive redundancy. Experiments are conducted three times.

B.3 Robust Training

We give the full results of robust training experiments on CIFAR-10 and ImageNet with confidence intervals in Tables 16 and 17. We use $\lambda = 0.5$ as the coefficient of the KL regularization term in the loss function in Section 3.2. The level of the noise (standard deviation) injected during the training is adjusted according to r and α values to be used at storage time since the noise at storage time has a standard deviation of $\frac{\sigma(w_{in})}{\alpha\sqrt{r}}$. It is seen from Tables 16 and 17 that robust training improves the robustness of the network against noise. We have also observed that the pruned network reaches higher accuracy after robust training compared to naive training without noise injection (an increase from 90.2% to 95.0% without retraining).

B.4 Robust Distillation

Knowledge distillation (KD) is a well-established NN compression method where a large teacher network is trained with $\mathcal{L}_t$ loss given by

$$\mathcal{L}_t = \mathbb{E}_{x,y\sim p_{data}}[-\log p_{w_t}(y|x)]$$

where w_t is the weights of the teacher network, and probability of each class i is the output of the high temperature softmax activation applied to logits:

$$y_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}.$$

where z_i is the logit for class i. Temperature $T > 1$ helps the output probabilities of the teacher network be softer. Using the same temperature, a smaller student network can be trained with the following student loss function $\mathcal{L}_s$:

$$\mathcal{L}_s = (1-\lambda)\mathbb{E}_{x,y\sim p_{data}}[-\log p_{w_s}(y|x)] + \lambda\mathbb{E}_x[D_{KL}(p_{w_t}(y|x)||p_{w_s}(y|x))]$$

where w_s is the weights of the student network. It has been shown that distilled student network achieves a test accuracy that a teacher network with the same architecture cannot achieve. In other words, a student network distilled from a teacher network performs comparable to a larger network. This suggests that knowledge distillation can be regarded as a promising compression method. In addition to compression, distillation has been shown to be an effective method for other desired NN attributes such as generalizability and adversarial robustness [55]. In this work, we define "robustness" as preserving a network's downstream classification accuracy when noise is added to the weights. This is achieved in part by robust training in the previous section where a trained network is robust to pruning and noise on the weights. Here, we present a student loss function that would make a (compressed) student network more robust to noise with no change in the teacher network training:

$$\mathcal{L}_s = (1-\lambda)\mathbb{E}_{x,y\sim p_{data}}[-\log p_{g(\hat{w}_s)}(y|x)] + \lambda\mathbb{E}_x[D_{KL}(p_{w_t}(y|x)||p_{g(\hat{w}_s)}(y|x))]$$

Table 18. Accuracy of ResNet-20 Distilled from ResNet-18 on CIFAR-10 when Weights are Perturbed by PCM

	Number of PCM cells	Teacher ResNet-18	Teacher ResNet-20	Student ResNet-20	Noisy Student ResNet-20	Add. Bits
No Noise	16	95.70	92.50	92.90	**93.00**	0
PCM+AP	3	16.23 (± 1.6)	48.38 (± 14.3)	73.38 (± 2.4)	**81.75** (± 1.5)	1
PCM+SP+AP	1	93.35 (± 0.5)	86.30 (± 1.5)	88.58 (± 0.4)	**90.65** (± 0.7)	2
	3	94.78 (± 0.2)	89.73 (± 0.2)	89.98 (± 0.3)	**91.33** (± 0.2)	
PCM+AP+Sens.	1	9.60(± 0.4)	29.68 (± 0.4)	38.18 (± 7.2)	**69.49** (± 3.7)	2
PCM+SP+AP+Sens.	1	93.36 (± 0.2)	88.40 (± 1.4)	88.96 (± 0.7)	**91.10** (± 0.3)	3
	3	94.90 (± 0.2)	89.92 (± 0.5)	90.44 (± 0.6)	**91.78** (± 0.2)	

SP: sign protection, AP: adaptive protection (adaptive mapping+sparsity-driven adaptive redundancy), Sens.: sensitivity-driven adaptive redundancy. During the distillation of noisy student, a Gaussian noise with N(0,0.01) is injected onto the weights. Experiments are conducted five times.

Table 19. Accuracy of ResNet-20 Distilled from ResNet-18 on CIFAR-10 when Weights are Perturbed by the Gaussian Noise

	Teacher ResNet-18	Teacher ResNet-20	Student ResNet-20	Noisy Student ResNet-18	Additional Bits
No Noise	95.70	92.50	92.90	**93.00**	0
N(0,0.01)+No Protection	82.90 (± 2.3)	86.44 (± 1.6)	89.10 (± 0.7)	**90.56** (± 0.4)	0
N(0,0.01)+SP+AP	95.60 (± 0.0)	92.30 (± 0.1)	92.66 (± 0.0)	**92.76** (± 0.1)	2
N(0,0.02)+No Protection	10.88 (± 1.2)	45.36 (± 8.8)	65.50 (± 7.0)	**77.78** (± 3.7)	0
N(0,0.02)+SP+AP	95.70 (± 0.0)	91.68 (± 0.4)	92.30 (± 0.1)	**92.36** (± 0.2)	2

SP: sign protection, AP: adaptive protection (adaptive mapping+sparsity-driven adaptive redundancy). During the distillation of noisy student, a Gaussian noise with N(0,0.01) is injected onto the weights. Experiments are conducted five times.

In the experiments, we use a temperature parameter of $T = 1.5$ and equally weight the contributions of the student network's cross entropy loss and the KL term ($\lambda = 0.5$). Similar to robust training, the noise level (standard deviation) during training is adjusted according to r and α values to be used at storage time. We give the full results on CIFAR-10 with confidence intervals in Tables 18 and 19 with weights perturbed by PCM cells and Gaussian noise, respectively. We also present the same set of experiments on MLP distilled from LeNet (on MNIST) in Tables 20 and 21. As in CIFAR-10 experiments, noise injection during distillation makes the student network more robust to both PCM and Gaussian noise in MNIST experiments.

Table 20. Accuracy of MLP Distilled from LeNet on MNIST when Weights are Perturbed by the PCM Cells

	Teacher LeNet	Teacher MLP (Student baseline)	Student MLP (No Noise)	Noisy Student MLP (with N(0,0.1))	Noisy Student MLP (with N(0,0.006))	Number of PCM cells	Additional Bits
No Noise	99.20	97.50	97.80	96.30	**97.30**	16	0
PCM+SP	98.94 (± 0.0)	91.86 (± 1.7)	87.46 (± 3.7)	**95.46**(± 0.1)	93.12 (± 1.4)	1	1
PCM+AP	98.60 (± 0.2)	92.76 (± 1.6)	95.40 (± 1.0)	95.78(± 0.2)	**96.04** (± 0.4)	1	1
PCM+SP+AP	99.20 (± 0.0)	97.04 (± 0.1)	97.46 (± 0.2)	96.12 (± 0.1)	**97.58** (± 0.1)	1	2
PCM+SP+AP+Sens.	99.20 (± 0.0)	97.20 (± 0.0)	97.72 (± 0.1)	96.14(± 0.1)	**97.80** (± 0.1)	1	3
PCM+AP+Sens.	98.88 (± 0.1)	95.04 (± 0.5)	97.10 (± 0.1)	95.92(± 0.1)	**97.46** (± 0.1)	1	3

SP: sign protection, AP: adaptive protection (adaptive mapping+sparsity-driven adaptive redundancy), Sens.: sensitivity-driven adaptive redundancy. Experiments are conducted five times.

Table 21. Accuracy of MLP Distilled from LeNet on MNIST when Weights are Perturbed by the Gaussian Noise

	Teacher LeNet	Teacher MLP (Student baseline)	Student MLP (No Noise)	Noisy Student MLP (with N(0,0.1))	Noisy Student MLP (with N(0,0.006))	Additional Bits
No Noise	99.20	97.50	97.80	96.30	**97.30**	0
N(0,0.1)+No Protection	22.60 (± 7.7)	77.72 (± 3.8)	58.90 (± 3.6)	**93.80** (± 0.4)	62.20 (± 3.5)	0
N(0,0.1)+SP+AP	99.22 (± 0.0)	95.92 (± 0.3)	95.96 (± 0.8)	95.82 (± 0.2)	**97.02** (± 0.2)	2
N(0,0.2)+No Protection	12.02 (± 3.0)	33.50 (± 3.4)	24.78 (± 6.3)	**74.82**(± 5.0)	**25.02**(± 2.8)	0
N(0,0.2+SP+AP)	99.18 (± 0.1)	90.36 (± 1.0)	86.30 (± 4.6)	**95.06**(± 0.3)	93.06 (± 2.1)	2
N(0,0.06)+No Protection	99.24 (± 0.0)	97.38 (± 0.1)	97.82 (± 0.1)	96.28 (± 0.0)	**97.86** (± 0.0)	0
N(0,0.06)+SP+AP	99.20 (± 0.0)	97.50 (± 0.0)	97.80 (± 0.0)	96.30 (± 0.0)	**97.90** (± 0.0)	2

SP: sign protection, AP: adaptive protection (adaptive mapping+sparsity-driven adaptive redundancy). Experiments are conducted five times.

ACKNOWLEDGMENTS

The authors would like to thank TSMC Corporate Research for the technical discussions; and Robert M. Radway and Pulkit Tandon for their constructive feedback.

REFERENCES

[1] Alessandro Achille, Giovanni Paolini, and Stefano Soatto. 2019. Where is the information in a deep neural network? *arXiv preprint arXiv:1905.12213* (2019).

[2] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2018. Scalable methods for 8-bit training of neural networks. In *Advances in Neural Information Processing Systems*. 5145–5153.

[3] Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems* 32 (2019).

[4] David Barber and Felix V. Agakov. 2003. The IM algorithm: A variational approach to information maximization. In *Advances in Neural Information Processing Systems*. None.

[5] Leighton Pate Barnes, Huseyin A. Inan, Berivan Isik, and Ayfer Özgür. 2020. rTop-k: A statistical estimation approach to distributed SGD. *IEEE Journal on Selected Areas in Information Theory* 1, 3 (2020), 897–907.

[6] Jonathan Binas, Daniel Neil, Giacomo Indiveri, Shih-Chii Liu, and Michael Pfeiffer. 2016. Precise neural network computation with imprecise analog devices. *arXiv preprint arXiv:1606.07786* (2016).

[7] Thomas Bird, Johannes Ballé, Saurabh Singh, and Philip A. Chou. 2021. 3D scene compression through entropy penalized neural representation functions. In *2021 Picture Coding Symposium (PCS)*. IEEE, 1–5.

[8] G. M. Bo, D. D. Caviglia, and M. Valle. 2000. An on-chip learning neural network. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, Vol. 4. 66–71. https://doi.org/10.1109/IJCNN.2000.860751

[9] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06).* Association for Computing Machinery, New York, NY, USA, 535–541. https://doi.org/10.1145/1150402.1150464

[10] G. W. Burr, S. Ambrogio, P. Narayanan, H. Tsai, C. Mackin, and A. Chen. 2019. Accelerating deep neural networks with analog memory devices. In *2019 China Semiconductor Technology International Conference (CSTIC).* 1–3. https://doi.org/10.1109/CSTIC.2019.8755642

[11] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine* 35, 1 (2018), 126–136.

[12] Yu Cheng, Felix X. Yu, Rogerio S. Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. 2015. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision.* 2857–2865.

[13] Kristy Choi, Kedar Tatwawadi, Aditya Grover, Tsachy Weissman, and Stefano Ermon. 2019. Neural joint source-channel coding. In *International Conference on Machine Learning.* PMLR, 1182–1192.

[14] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. 2020. Universal deep neural network compression. *IEEE Journal of Selected Topics in Signal Processing* (2020).

[15] Yann Le Cun, John S. Denker, and Sara A. Solla. 1990. *Optimal Brain Damage.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 598–605.

[16] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems.* 1223–1231.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR09.*

[18] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.

[19] Yuan Du, Li Du, Xuefeng Gu, Jieqiong Du, X. Shawn Wang, Boyu Hu, Mingzhe Jiang, Xiaoliang Chen, Subramanian S. Iyer, and Mau-Chung Frank Chang. 2018. An analog neural network computing engine using CMOS-compatible charge-trap-transistor (CTT). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 10 (2018), 1811–1819.

[20] Gintare Karolina Dziugaite, Gabriel Arpino, and Daniel M. Roy. 2018. Towards generalization guarantees for SGD: Data-dependent PAC-bayes priors. (2018).

[21] J. H. Engel, S. B. Eryilmaz, S. Kim, M. BrightSky, C. Lam, H. Lung, B. A. Olshausen, and H. P. Wong. 2014. Capacity optimization of emerging memory systems: A Shannon-inspired approach to device characterization. In *2014 IEEE International Electron Devices Meeting.* 29.4.1–29.4.4.

[22] Omobayode Fagbohungbe and Lijun Qian. 2020. Benchmarking inference performance of deep learning models on analog devices. *arXiv preprint arXiv:2011.11840* (2020).

[23] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. 2020. Training with quantization noise for extreme model compression. (2020).

[24] Scott W. Fong, Christopher M. Neumann, and H.-S. Philip Wong. 2017. Phase-change memory-towards a storage-class memory. *IEEE Transactions on Electron Devices* 64, 11 (2017), 4374–4385.

[25] Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *International Conference on Learning Representations (ICLR)* (2019).

[26] Roger Grosse and James Martens. 2016. A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning.* 573–582.

[27] Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient DNNs. In *Advances in Neural Information Processing Systems.* 1379–1387.

[28] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *International Conference on Learning Representations (ICLR)* (2016).

[29] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems.* 1135–1143.

[30] Babak Hassibi, David G. Stork, Gregory Wolff, and Takahiro Watanabe. 1993. Optimal brain surgeon: Extensions and performance comparisons. In *Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS'93).* San Francisco, CA, USA, 263–270.

[31] Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. 2019. Minimal random code learning: Getting bits back from compressed model parameters. In *International Conference on Learning Representations (ICLR).*

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 770–778.

[33] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*. http://arxiv.org/abs/1503.02531

[34] Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. 2020. Characterising bias in compressed models. *arXiv preprint arXiv:2010.03058* (2020).

[35] Berivan Isik. 2021. Neural 3D scene compression via model compression. *arXiv preprint arXiv:2105.03120* (2021).

[36] Berivan Isik, Philip Chou, Sung Jin Hwang, Nicholas Johnston, and George Toderici. 2021. LVAC: Learned volumetric attribute compression for point clouds using coordinate based networks. *Frontiers in Signal Processing* (2021), 65.

[37] Berivan Isik, Albert No, and Tsachy Weissman. 2021. Rate-distortion theoretic model compression: Successive refinement for pruning. *arXiv preprint arXiv:2102.08329* (2021).

[38] Berivan Isik, Francesco Pase, Deniz Gunduz, Tsachy Weissman, and Zorzi Michele. 2023. Sparse random networks for communication-efficient federated learning. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=k1FHgri5y3-

[39] Berivan Isik and Tsachy Weissman. 2022. Learning under storage and privacy constraints. In *2022 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 1844–1849.

[40] Berivan Isik, Tsachy Weissman, and Albert No. 2022. An information-theoretic justification for model pruning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 3821–3846.

[41] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[42] V. Joshi, M. Le Gallo, Simon Haefeli, I. Boybat, S. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian, and E. Eleftheriou. 2020. Accurate deep neural network inference using computational phase-change memory. *Nature Communications* 11 (2020).

[43] Soroosh Khoram and Jing Li. 2018. Adaptive quantization of neural networks. In *International Conference on Learning Representations*.

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

[45] Souvik Kundu, Mahdi Nazemi, Peter A. Beerel, and Massoud Pedram. 2021. DNR: A tunable robust pruning framework through dynamic network rewiring of DNNs. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 344–350.

[46] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[47] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[48] Yann LeCun, Corinna Cortes, and C. J. Burges. 2010. MNIST handwritten digit database. (2010).

[49] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. 2018. SNIP: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340* (2018).

[50] Christos Louizos, Karen Ullrich, and Max Welling. 2017. Bayesian compression for deep learning. *arXiv preprint arXiv:1705.08665* (2017).

[51] James Martens. 2014. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193* (2014).

[52] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*. Springer, 405–421.

[53] S. R. Nandakumar, Irem Boybat, Vinay Joshi, Christophe Piveteau, Manuel Le Gallo, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou. 2019. Phase-change memory models for deep learning training and inference. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 727–730. https://doi.org/10.1109/ICECS46596.2019.8964852

[54] Deniz Oktay, Johannes Ballé, Saurabh Singh, and Abhinav Shrivastava. 2019. Scalable model compression by entropy penalized reparameterization. *arXiv preprint arXiv:1906.06624* (2019).

[55] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 582–597.

[56] Francesco Pase, Berivan Isik, Deniz Gunduz, Tsachy Weissman, and Michele Zorzi. [n. d.]. Efficient federated random subnetwork training. In *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*.

[57] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* (2018).

[58] Brandon Reagan, Udit Gupta, Bob Adolf, Michael Mitzenmacher, Alexander Rush, Gu-Yeon Wei, and David Brooks. 2018. Weightless: Lossy weight encoding for deep neural network compression. In *International Conference on Machine Learning*. 4324–4333.

[59] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[60] Alexandre Schmid, Yusuf Leblebici, and D. Mlynek. 2000. Mixed analogue-digital artificial-neural-network architecture with on-chip learning. *Circuits, Devices and Systems, IEE Proceedings* - 146 (01 2000), 345–349. https://doi.org/10.1049/ip-cds:19990685

[61] Vikash Sehwag, Shiqi Wang, Prateek Mittal, and Suman Jana. 2020. Hydra: Pruning adversarially robust neural networks. *Advances in Neural Information Processing Systems (NeurIPS)* 7 (2020).

[62] Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review* 5, 1 (2001).

[63] Sidak Pal Singh and Dan Alistarh. 2020. WoodFisher: Efficient second-order approximations for model compression. *arXiv preprint arXiv:2004.14340* (2020).

[64] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 6105–6114.

[65] Thuy Van Nguyen, Aria Nosratinia, and Dariush Divsalar. 2012. The design of rate-compatible protograph LDPC codes. *IEEE Transactions on Communications* 60, 10 (2012), 2841–2850.

[66] S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinč, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand, D. Marpe, and W. Samek. 2020. DeepCABAC: A universal compression algorithm for deep neural networks. *IEEE Journal of Selected Topics in Signal Processing* 14, 4 (2020), 700–714.

[67] H.-S. Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P. Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E. Goodson. 2010. Phase change memory. *Proc. IEEE* 98, 12 (2010), 2201–2227.

[68] J. Y. Wu, Y. S. Chen, W. S. Khwa, S. M. Yu, T. Y. Wang, J. C. Tseng, Y. D. Chih, and Carlos H. Diaz. 2018. A 40nm low-power logic compatible phase change memory technology. In *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 27–6.

[69] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. 2020. Self-training with noisy student improves ImageNet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10687–10698.

[70] Sean I. Young, Wang Zhe, David Taubman, and Bernd Girod. 2020. Transform quantization for CNN compression. *arXiv preprint arXiv:2009.01174* (2020).

[71] Ryan Zarcone, Dylan Paiton, Alex Anderson, Jesse Engel, H. S. Philip Wong, and Bruno Olshausen. 2018. Joint source-channel coding with neural networks for analog data compression and storage. In *2018 Data Compression Conference*. IEEE, 147–156.

[72] Ryan V. Zarcone, Jesse H. Engel, S. Burc Eryilmaz, Weier Wan, SangBum Kim, Matthew BrightSky, Chung Lam, Hsiang-Lan Lung, Bruno A. Olshausen, and H.-S. Philip Wong. 2020. Author correction: Analog coding in emerging memory systems. *Scientific Reports* 10, 1 (August 2020), 13404. https://doi.org/10.1038/s41598-020-70121-y

[73] Xin Zheng, Ryan Zarcone, Dylan Paiton, Joon Sohn, Weier Wan, Bruno Olshausen, and H.-S. Philip Wong. 2018. Error-resilient analog image storage and compression with analog-valued RRAM arrays: An adaptive joint source-channel coding approach. In *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 3–5.

[74] Chuteng Zhou, Prad Kadambi, Matthew Mattina, and Paul N. Whatmough. 2020. Noisy machines: Understanding noisy neural networks and enhancing robustness to analog hardware errors using distillation. *arXiv preprint arXiv:2001.04974* (2020).

[75] Chuteng Zhou, Quntao Zhuang, Matthew Mattina, and Paul N. Whatmough. 2021. Information contraction in noisy binary neural networks and its implications. *arXiv preprint arXiv:2101.11750* (2021).

[76] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. 2018. Non-vacuous generalization bounds at the ImageNet scale: A PAC-Bayesian compression approach. *arXiv preprint arXiv:1804.05862* (2018).

Received 17 March 2022; revised 2 September 2022; accepted 3 February 2023