MEMTI: optimizing on-chip non-volatile storage for visual multi-task inference at the edge

Marco Donato[1], Lillian Pentecost[1], David Brooks[1,2], and Gu-Yeon Wei[1]

[1]Harvard University
[2]Facebook

Abstract—The combination of specialized hardware and embedded non-volatile memories (eNVM) holds promise for energy-efficient DNN inference at the edge. However, integrating DNN hardware accelerators with eNVMs still presents several challenges. Multi-level programming is desirable for achieving maximal storage density on chip, but the stochastic nature of eNVM writes makes them prone to errors and further increases the write energy and latency. We present MEMTI, a memory architecture that leverages a multi-task learning technique for maximal reuse of DNN parameters across multiple visual tasks. We show that by retraining and updating only 10% of all DNN parameters, we can achieve efficient model adaptation across a variety of visual inference tasks. The system performance is evaluated by integrating the memory with the open-source NVIDIA Deep Learning Architecture (NVDLA).

Index Terms—DNN accelerators, edge computing, multi-task learning, non-volatile memories

1 Introduction

In recent years, deep neural networks (DNNs) have become essential to tasks across application domains, including image recognition and detection, language processing, and translation. This increase in popularity, together with the continued proliferation of low-power embedded devices, has motivated the design of DNN-specific hardware accelerators [1]. While many energy-efficient DNN hardware implementations have been proposed, a major challenge remains: the large memory requirement to store DNN parameters. Although entirely on-chip storage would guarantee better inference performance, limited on-chip SRAM capacity inevitably leads to reliance on costly off-chip memory accesses to DRAM.

Embedded non-volatile memories (eNVMs) provide higher density than SRAM and can ameliorate the need for power-hungry DRAM storage. However, the benefits of eNVMs come at the cost of larger write energy and write latency. Moreover, limited eNVM write endurance is an obstacle to the adoption of certain technologies if DNN parameter values require frequent updates. For instance, embedded devices for robotics or augmented reality applications often required a combination of multiple inference tasks, including image classification, object detection, and action recognition. These cases highlight the need for scalable solutions that can flexibly accommodate DNN parameters for multiple tasks.

We present a DNN model and memory co-design solution that leverages a machine learning technique described in Section 2 to reduce eNVM writes, while enabling systems to efficiently perform multiple inference tasks. Maximizing the reuse of the learned parameters across different DNN-dependent vision tasks without re-training enforces the assumption of infrequent writes: parameters shared by multiple tasks are trained and written only once, and therefore are highly suitable for eNVM storage; in contrast, the remaining parameters can be re-trained to accommodate new inference tasks, and stored in SRAM. In addition to the storage density benefits, we evaluate how the process of re-training specific parameters can be used to recover from accuracy loss due to the adoption of denser, fault-prone multi-level eNVM storage. This paper provides the following contributions:

- Leverage residual adapters to optimize parameter storage in dense eNVMs;
- Evaluate application accuracy with quantization and MLC RRAM faults when a majority of DNN parameters is shared across inference tasks;
- Quantify the system-level performance and energy advantages of a multi-task-enabled deep learning architecture (NVDLA) integrated with optimized eNVM solutions.

2 DNN and memory co-design

Generalizing deep learning architectures to enable different application domains and more varied inference tasks serves as a way of supporting more powerful and versatile models. For example, [2] combines several building blocks for translation, speech, and visual inference that can be trained on all desired tasks simultaneously or on each task separately. In either case, however, introducing new inference tasks would require updating the entire set of model parameters. Other works have leveraged the concept of transfer learning to improve the performance of a single DNN on different datasets. These approaches are based on the observation that many visual inference tasks share low-level features, such as edge and shape detection, in the front-end layers, and become more task-specific as the computation moves closer to the classification layers. However, in order to preserve inference accuracy, transfer learning approaches either share only a limited number of front-end layers or

TABLE 1
Summary of dataset characteristics, and maximum training accuracy for the model trained entirely from scratch on each dataset or using residual adapters on a pre-trained network. Pre-trained shared parameters on ImageNet, with 67.65% accuracy.

	Dataset				
	cifar100	aircraft	daimlerpedcls	gtsrb	ucf101
# images	50K	7K	30K	40K	9K
# classes	100	100	2	43	101
Full model	72.78%	40.98%	99.88%	99.97%	73.77%
Only adapters	79.61%	43.8%	99.51%	99.94%	73.16%
Parameters overhead	10.4%	10.4%	10.1%	10.2%	10.4%
Training speed-up	4×	2×	1.35×	3.23×	4.74×

fine-tune parameters by re-training the transferred features from one inference task to another [3]. A recent proposal applies transfer learning to create a synthesizable fixed-parameter feature extractor [4]. However, hard-wiring the feature extractor in logic prevents from fine-tuning the parameters, limiting the amount of cross-task weight sharing.

While all these techniques enable a single DNN model to perform different inference tasks, they still require updating a considerable portion of parameters to achieve maximum adaptation. We pursue a specific transfer learning technique for which the learned parameters can be generalized across multiple vision inference tasks by maximizing DNN parameter reuse and enabling efficient inference on embedded devices. The high degree of DNN parameters reuse reduces memory traffic requirements, which makes non-volatile memories a compelling solution for retaining shared parameters on-chip without incurring costs associated with frequent memory writes.

2.1 Multi-task learning model

Our design is based on the DNN architecture presented in [5], which uses residual adapter modules as a way to parameterize a generic ResNet network. These parametric modules are themselves residual blocks which use 1×1 filters and skip connection. In this setting, the number of domain-specific parameters, which comprises adapter filters, batch normalization, and fully-connected classifier parameters, can be reduced to roughly 10% of the total model size. For our experiments, we integrate the residual adapter modules in a ResNet26 network.

The baseline network is pre-trained on ImageNet, which is standard practice in transfer learning and model fine-tuning techniques. The pre-trained version for ImageNet achieves top-1 accuracy of 67.65%. The ResNet26 weight parameters obtained during pre-training are the backbone of this multi-task inference system as they are reused for running inference on any additional visual task. The degree of adaptation is tested against five datasets which have been selected to be representative of popular image processing tasks including classification (cifar100, aircraft), object detection (German Traffic Signs, Daimler pedestrian classification), and action recognition (UCF101 Dynamic Images).

Table 1 summarizes the best accuracy in the case of the model being either trained entirely from scratch or only for the task-specific parameters. As anticipated, for all datasets, the adapters overhead is around 10%. The accuracy of the network trained using adapters is always better than or comparable to training the entire network independently for each dataset. In addition, we observe that the modified model converges to the best accuracy in fewer training epochs, which results in training speed-up reported in Table 1.

2.2 Non-volatile memory technologies

The landscape of non-volatile memories includes a wide range of emerging technologies [6]. These memories are generally characterized by high energy efficiency and high storage density, which can be further increased by programming multiple levels in a single cell. We label this storage solution as multi-level cell (MLC) storage, in contrast to single-level cell (SLC) storage, for which each eNVM cell stores a single binary value. In this work, we focus on a specific eNVM implementation, namely RRAM. Various implementations such as phase-change memories (PCM), embedded Flash (eFlash), or ferroelectric memories (FeRAM) can also be used for MLC storage. On the other hand, STT magnetic memories (STT-MRAM), while having the best write and read performance [6], are not a suitable candidate because compelling MLC implementations with comparable density have not been demonstrated to date.

There are alternative implementations with varying advantages and limitations. For example, the storage requirements for the DNN architecture we are leveraging could be met by read-only memories (ROM) as well. ROMs ensure the best density for storing the shared parameters, however, they also require configuring the network at fabrication time, which makes the design less scalable and cost-effective. One-time programmable (OTP) memories such as anti-fuse, while being amenable to post-fabrication configuration, are far less dense than other memory solutions, even when compared to SRAM [7]. Previous work has investigated how threshold voltage shifts induced by hot-carrier injection in standard high-k transistors could be used as non-volatile memories [8]. Moreover, recent work has shown how eNVMs implemented with this approach can be used for on-chip MLC weight storage for DNN accelerators [9]. The same behavior has been demonstrated on a variety of technologies, including bulk, silicon-on-insulator, and FinFET devices. A major limitation for these eNVMs is the long write latency, which falls in the range of milliseconds.

2.3 MEMTI: Memory system for Energy-efficient Multi-Task Inference

In order to complement the properties of residual adapter networks and dense MLC RRAM storage, we propose MEMTI, a Memory system for Efficient Multi-Task Inference. A large fraction of the parameters in a residual adapter network is shared across multiple applications, and can be efficiently stored in MLC RRAM, while application-specific parameters can be stored in SRAM. By partitioning on-chip memory area between RRAM and SRAM, we achieve the best trade-off between storage density for the shared parameters and fast and energy-efficient updates for the task-specific parameters (SRAM). Off-chip DRAM stores multiple sets of task-specific parameters. The resulting memory

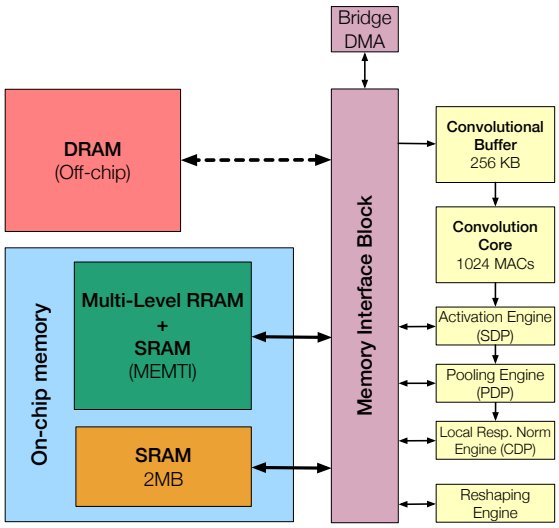

Fig. 1. NVDLA system diagram, with additional optional interface to multi-level-cell RRAM for on-chip weight storage.

hierarchy takes advantage of RRAM non-volatility for intermittent operation by powering down the system between inferences. In this scenario, only a small portion of the model parameters must be written to SRAM during power up or task switching. Moreover, storing task-specific parameters in more robust memory allows us to mask MLC RRAM faults via retraining, as shown in Section 4.

3 Evaluation Framework

To evaluate the proposed memory architecture, we quantify the impact of RRAM fault characteristics and MLC encoding on inference accuracy, memory architecture and array properties, and system-level performance. The fault model is derived from previous work integrating eNVM device and circuit-level fault characteristics with DNNs evaluation frameworks to allow for extensive memory and DNN co-design space exploration [9]. We model MLC RRAM faults based on stochastic level distribution which arise from the random nature of memristors programming. When multiple levels are programmed in a single RRAM cell, the distributions overlap can be used to extrapolate the read fault probabilities for each level.

The resulting error map is then integrated in a DNN evaluation framework to simulate the impact MLC RRAM faults on inference accuracy. The level distributions are extrapolated from measured MLC RRAM characteristics [10]. We use a version of the residual adapter architecture implemented in PyTorch to evaluate the DNN accuracy under different storage schemes. The existing implementation is modified by adding transform functions that manipulate the weight parameters value according to different multi-level encoding and compression techniques.

Based on the MLC RRAM fault probabilities, we sample the value of the stored weight matrix based on a predefined multi-level encoding configuration to evaluate the impact on the model accuracy. In addition, we improve the fault model by including the effects of the sensing circuitry on the read error probability.

The corresponding framework is used to drive the design towards a solution that would minimize the on-chip memory footprint without increasing the inference error. After identifying the best MLC encoding without loss in accuracy, we perform a memory design space exploration using a modified version of NVSim [11]. Once again, we consider the contribution of sense amplifiers to area, energy and performance of the memory array. Reading back the stored value requires converting the programmed analog level to a binary word, and can be done using parallel sensing or sequential sensing schemes. Parallel sensing is similar to using a flash ADC, and requires each bitline to have dedicated sense amplifiers for each possible stored level. Sequential sensing uses a single sense amplifier and recovers the stored binary word iteratively for each bit. While sequential sensing reduces the overall number of sense amplifiers, we noticed that implementing parallel sensing with small sense amplifiers does not incur an excessive area penalty. The impact of off-chip memory accesses is quantified using a model of LPDDR4 DRAM. Power and performance estimates are derived assuming a power consumption of $200\mathrm{mW}$ at a $1\mathrm{GHz}$ operating frequency. Finally, we integrate the resulting memory hierarchy with a proven CNN accelerator architecture developed by NVIDIA (NVDLA), which, combined with NVSim results and DRAM estimates, allows us to evaluate the system energy and performance for different application scenarios.

4 Model compression and training techniques

In this Section, we explore the trade-offs between different storage techniques and model accuracy. In particular, we look at the combined effect of circuit-level optimization (RRAM MLC encoding) and reducing DNN model size (quantization and pruning). Quantizing the whole model using a fixed point encoding with 2 bits for sign and integer and 6 bits for the fractional part achieves 80.3% accuracy on cifar100. We highlight some key insights by considering three examples. For all three cases, we take advantage of the residual adapter ability to compensate for accuracy loss associated with the faults in MLC RRAM and due to reduced DNN weight precision. We demonstrate that by fine-tuning only the task-specific parameters, the DNN learns variability-induced errors affecting the parameters stored in RRAM, and this helps maintain inference accuracy closer to the baseline value. The benefit of this approach is that the impact of faults in the non-volatile memory can be minimized without introducing any additional circuit overhead.

Our first example shows the implications of storing both shared and task-specific parameters on 3 bits/cell MLC RRAM. For 8-bit weights, we can reduce the effective fault rate for the sign and integer values by spacing the levels as described by the non-uniform encoding technique presented in [9]. In addition, we also prune the shared parameters, which has also been shown to help mask MLC errors in non-volatile memories [9]. We observe that storing all weights for cifar100 in MLC RRAM results in an average accuracy over 100 trials of 28.34%, and retraining the residual parameters raises the accuracy to just 56.98%. These results

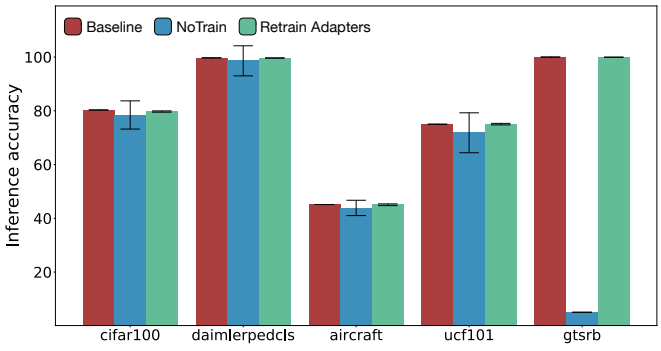

Fig. 2. Accuracy for different datasets when the weights are stored using MEMTI. The results before and after task-specific parameters retraining are compared to the baseline model accuracy for the compressed, error-free model. Each bar shows the mean accuracy and standard deviation over 100 random trials.

highlight the importance of protecting the value of the task-specific parameters from errors. Several approaches can be used to mitigate the impact of MLC faults on accuracy. Error correction codes (ECC) are an established solution for improving the reliability of fault-prone MLC eNVMs and they can be adopted with reasonable overheads in terms of additional circuitry [12]. Alternatively, the memory density can be traded off with resiliency to faults by adopting SLC RRAM for storing the task specific parameters. Neither solution solves the issues related to eNVM write endurance, and these options represent possible extensions to working with residual adapters for multi-task scenarios, rather than concrete alternatives.

For these reasons, MEMTI proposes a hybrid memory architecture in which shared and task-specific parameters are split between RRAM and SRAM. Starting from the same quantization and MLC configuration as in the previous example, we show an average accuracy of 79.72% after retraining the residual parameters if residual parameters are stored securely in SRAM. In fact, this retraining strategy can completely mask the impact of storing weights in MLC RRAM; even for the worst case accuracy degradation when storing shared parameters in RRAM (36.91%), re-training and securely storing the residual adapter parameters results in an accuracy of 79.24%, consistent with baseline accuracy. Motivated by this result, we propose leveraging this technique to achieve even more aggressively dense storage by reducing the number of bits for the shared weights to 6. This configuration uses just 2 RRAM cells per weight. In this case, training the residual adapters results in an average accuracy of 79.05%. The worst-case accuracy before training the adapters is 2.04%, and can be recovered up to 77.57% after training. Figure 2 shows the same trend in terms of the nominal baseline accuracy, accuracy before re-training, and accuracy after retraining of the task-specific parameters for all the different datasets introduced in Section 2 .

5 System-level characterization

As shown in Figure 1, the baseline NVDLA system comprises a convolutional core with 1024 MAC units fed by a convolutional buffer and supplemented by several additional computational units for pooling and data transformation operations. NVDLA also supports a memory interface block and DMA that fetches model weights per layer from off-chip DRAM and leverages on-chip SRAM (2MB) to buffer inputs and intermediate results of computation between layers in the DNN. We flexibly integrate MEMTI with the NVDLA performance model as an additional memory interface to leverage for model weights, either in addition to or in place of fetching parameters from off-chip DRAM.

For a competitive multi-task inference application, we evaluate the performance, energy, and area for the NVDLA system when executing three inferences per input frame using three representative visual tasks, namely image classification (cifar100), object detection (gtsrb), and action recognition (UCF101). This series of tasks computed per input frame would be appropriate, for example, for an autonomous vehicle or a drone processing sensory data to understand and interact with the surrounding environment. For this application, we set the target operating frequency to 30 frames per second (FPS), or 90 inference tasks per second, which satisfies a breadth of applications. Both NVSim and NVDLA results are extrapolated for a system manufactured using a $22\mathrm{nm}$ technology node.

DRAM-based design: As a baseline case we assume that the accelerator is continuously processing input frames and fetching both shared and task-specific parameters from off-chip DRAM for each layer's computation. This DRAM-only, always-on operation consumes a total power of $493\mathrm{mW}$ and a peak performance of $749\mathrm{FPS}$. The estimated power includes datapath, DRAM refresh, and on-chip SRAM leakage. At this stage, the on-chip SRAM is exclusively used for storing the input features and intermediate values. We compute the energy per frame at peak performance to be $1.17\mathrm{mJ}$.

Provisioning for on-chip SRAM: We first show the case in which we allocate enough on-chip SRAM to store the entire set of parameters for a single task. For a system designed to run a single inference task, having the option of storing all the network parameters on chip allows to reduce the memory access energy by $40\times$. This result demonstrates the strong impact of off-chip memory access on the entire system energy. These high energy savings are however impractical to realize with SRAM since for a $22\mathrm{nm}$ technology node, we estimate a total area of $6.55\mathrm{mm}^2$. Moreover, when we consider the full system energy in the multi-task scenario described above, the periodic parameter updates and SRAM leakage power reduce the energy savings to $0.64\times$.

Improving storage density with eNVM: As a first step towards reducing both power consumption and memory footprint we consider storing the weights on chip using MLC RRAM. Without applying any DNN-level optimization, a multi-task operation would still require updating all the model parameters stored in RRAM when switching to different inference tasks. While this type of operation has a clear downside dictated by the RRAM write endurance, our system level evaluation exposes other limitations. Although the overall leakage power and on-chip memory area can be reduced to $298\mathrm{mW}$ and $0.347\mathrm{mm}^2$ respectively, the energy per inference increases to $14.58\mathrm{mJ}$. This is caused by the combined effect of RRAM write energy and latency. These examples highlight the need for a solution capable of balancing on-chip memory density and write perfor-

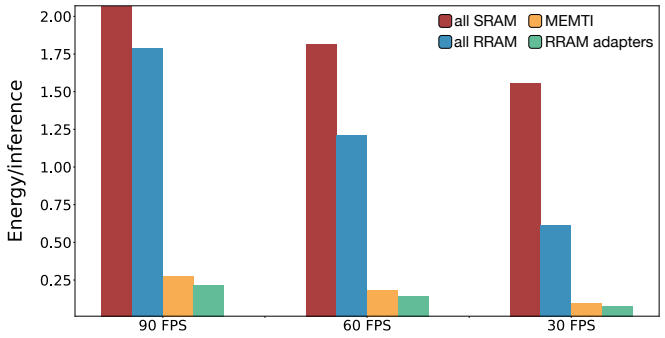

Fig. 3. Energy vs FPS for the different design configurations normalized to the DRAM baseline. The power savings of the RRAM-based design for higher frame rates is exacerbated by the frequent RRAM writes, making the design less efficient than the DRAM-based baseline. On the other hand, the energy per inference for MEMTI and RRAM adapters is strictly better than the baseline.

mance. *MEMTI for intermittent operation:* MEMTI removes the memory write costs by replacing the RRAM portion storing the task-specific parameters with SRAM. This is possible thanks to the adoption of residual adapters in the DNN network. For the resulting design, the total system power is $362\mathrm{mW}$ and the energy per inference is $1.56\mathrm{mJ}$, which is comparable with the baseline result. Isolating the costs associated with weight storage emphasizes the benefits introduced by MEMTI: power consumption is reduced by $3.9\times$, with an area overhead of $1.16\mathrm{mm}^2$. The resulting peak performance is $429\mathrm{FPS}$, well above the application requirements. Intermittent operation is where MEMTI truly stands out by taking advantage of the non-volatility of RRAM. In this scenario, we fix the operation at $30\mathrm{FPS}$ and power down the system between frames, which reduces the energy per inference by $10.65\times$.

A RRAM-based specialized design: Alternatively, we consider the case specifically tailored for the three chosen tasks. Using residual adapters still reduces the weights storage requirements by a factor of $2.3\times$. However, based on the results from Section 4, we use SLC RRAM to store the task-specific parameters and preserve inference accuracy. Therefore, we store the entire set of parameters in MLC and SLC RRAM. Removing the need for additional SRAM reduces the overall power to $343\mathrm{mW}$, and the area to $1\mathrm{mm}^2$. Allocating enough memory for storing all the parameters on chip increases the energy savings compared to the baseline by $13.6\times$, making this design the most area and energy efficient. Nonetheless, MEMTI maintains the advantage in terms of flexibility and robustness to RRAM errors thanks to ease of reprogrammability for the task-specific parameters, for which the memory capacity is determined only by the network structure and therefore is independent from the breadth of tasks considered in a specific application.

Figure 3 shows the relationship between FPS and energy per inference normalized to the DRAM case. The all SRAM and all RRAM configurations are heavily penalized by the inability of efficiently implement a multi-task inference system. On the other hand, a co-design of the memory and DNN model using residual adapters shows much higher energy savings compared to the baseline. Table 2 summarizes the results at 30 FPS for the different configuration cases.

TABLE 2
Summary of power, performance and area for the four design configurations considered in this work. The energy savings are normalized to the all DRAM configuration for the intermittent multi-task operation over three tasks running at 30 FPS. On-chip RRAM shows the physical memory capacity (i.e. number of cells).

	Power [mW]	Max FPS	WMem Area [mm^2]	Saved energy	On-chip SRAM	On-chip RRAM
all DRAM	493	749	–	1×	2MB	–
all SRAM	634	485	6.55	0.64×	8.5MB	–
all RRAM	298	47	0.347	1.62×	2MB	2.2MB
MEMTI	344	429	1.16	10.65×	2.7MB	2MB
RRAM adapters	301	396	1	13.6×	2MB	4MB

6 Conclusion

With the increasing adoption of DNN hardware accelerators for edge devices, there is a growing need for scalable design approaches that provide flexible and cost-effective implementations. In evaluating the performance of different memory solutions integrated with a DNN hardware accelerator, we show that technological improvements alone do not always lead to the most optimized design, especially in the context of multi-task inference. A co-design approach that leverages the properties of emerging memory technologies and deep neural network models allows to achieve both energy efficiency and flexibility. With this in mind, we present MEMTI as a methodology for enabling energy-efficient multi-task inference on edge devices, while reducing the cost of non-volatile memory writes. In addition, training the task-specific parameters based on the memory characteristics allows recovery of accuracy lost due to fault-prone eNVM storage.

References

[1] B. Reagen, R. Adolf, P. Whatmough, G.-Y. Wei, and D. Brooks, "Deep Learning for Computer Architects," *Synth. Lect. Comput. Archit.*, vol. 12, pp. 1–123, aug 2017.

[2] L. Kaiser *et al.*, "One model to learn them all," *CoRR*, vol. abs/1706.05137, 2017.

[3] J. Yosinski *et al.*, "How transferable are features in deep neural networks?," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), pp. 3320–3328, MIT Press, 2014.

[4] P. N. Whatmough, C. Zhou, P. Hansen, S. K. Venkataramanaiah, J. sun Seo, and M. Mattina, "Fixynn: Efficient hardware for mobile computer vision via transfer learning," 2019.

[5] S.-A. Rebuffi *et al.*, "Efficient parametrization of multi-domain deep neural networks," 2018.

[6] D. C. Daly, L. C. Fujino, and K. C. Smith, "Through the looking glass - the 2018 edition: Trends in solid-state circuits from the 65th isscc," *IEEE Solid-State Circuits Magazine*, vol. 10, pp. 30–46, winter 2018.

[7] S. H. Kulkarni *et al.*, "A 32nm high-k and metal-gate anti-fuse array featuring a 1.01m21t1c bit cell," in *2012 Symposium on VLSI Technology (VLSIT)*, pp. 79–80, June 2012.

[8] F. Khan *et al.*, "The Impact of Self-Heating on Charge Trapping in High-k-Metal-Gate nFETs," *IEEE Electron Device Lett.*, 2016.

[9] M. Donato *et al.*, "On-chip deep neural network storage with multi-level envm," in *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pp. 169:1–169:6, 2018.

[10] L. Zhao, H.-Y. Chen, S.-C. Wu, Z. Jiang, S. Yu, T.-H. Hou, H. . P. Wong, and Y. Nishi, "Improved multi-level control of rram using pulse-train programming," in *Proceedings of Technical Program - 2014 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, pp. 1–2, April 2014.

[11] X. Dong *et al.*, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 2012.

[12] "What types of ECC should be used on flash memory?," Cypress Semiconductor, 2017.