DECO: Dynamic Energy-aware Compression and Optimization for In-Memory Neural Networks

Rebati Gaire†, Sepehr Tabrizchi†, Deniz Najafi‡, Shaahin Angizi‡ and Arman Roohi†

†School of Computing, University of Nebraska–Lincoln, Lincoln, NE, USA

‡Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA

aroohi@unl.edu, shaahin.angizi@njit.edu

2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS) | 979-8-3503-8717-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MWSCAS60917.2024.10658771

***Abstract*—This paper introduces DECO, a framework that combines model compression and processing-in-memory (PIM) to improve the efficiency of neural networks on IoT devices. By integrating these technologies, DECO significantly reduces energy consumption and operational latency through optimized data movement and computation, demonstrating notable performance gains on CIFAR-10/100 datasets. The DECO learning framework significantly improved the performance of compressed network modules derived from MobileNetV1 and VGG16, with accuracy gains of 1.66% and 0.41%, respectively, on the intricate CIFAR-100 dataset. DECO outperforms the GPU implementation by a significant margin, demonstrating up to a two-order-of-magnitude increase in speed based on our experiment.**

***Index Terms*—Model compression, feature extraction, processing in-memory**

I. Introduction

The Internet of Things (IoT) proliferation has ushered in an era of pervasive connectivity and data generation. IoT devices deployed across smart homes, factories, wearables, and countless other domains offer tremendous potential for real-time monitoring, intelligent automation, and data-driven insights. Deep learning (DL) is expected to revolutionize IoT applications, enabling capabilities such as predictive maintenance, anomaly detection, and personalized user experiences. However, the integration of DL models with IoT faces significant hurdles. Resource-constrained devices, limited network bandwidth, and the need for low-latency decision-making create a bottleneck for deploying complex DL algorithms at the edge. The explosive growth of deep neural networks (DNNs), fueled by their success in various domains, has led to increasingly complex architectures featuring millions or even billions of parameters. This complexity directly translates into high computational costs, extensive memory requirements, and substantial energy consumption, all of which conflict with the realities of IoT environments. To bridge this gap, researchers and practitioners have turned to software and hardware-intensive approaches to optimize DL's efficiency. Model compression [1]–[3] is a key software-based optimization strategy to reduce the size and computational demands of DNNs. Techniques like pruning, quantization, knowledge distillation, and network architecture design seek to create smaller, more efficient models while preserving accuracy. From the hardware perspective, processing-in-memory (PIM) architectures offer a paradigm shift. By integrating computational capabilities directly into or near memory, PIM aims to minimize data movement, a significant source of latency and energy consumption in traditional systems [4]. While both model compression and PIM offer substantial benefits, they also have limitations. Model compression often involves trade-offs between compression rates and accuracy. PIM, especially in its more radical forms, can introduce hardware complexity and require new programming and system design paradigms.

In this paper, we propose DECO, a novel approach that strategically combines elements of model compression and PIM to improve the efficiency of deep learning on IoT devices. Our method orchestrates a shared feature extractor across various neural network configurations, each distinguished by tailored model compression and distinct computational profiles, to cater to the dynamic energy and performance constraints of edge computing environments. We demonstrate its effectiveness through a rigorous evaluation, achieving noteworthy improvements in model efficiency and accuracy, as evidenced by the performance gains in classification tasks using the CIFAR-10 and CIFAR-100 datasets [5]. Our contribution unlocks the potential for more widespread adoption of DL adoption in IoT. Enabling intelligent analysis directly at the edge paves the way for faster, more responsive, and privacy-preserving IoT applications. Industries ranging from manufacturing to healthcare stand to benefit from this increased efficiency, leading to new levels of automation, optimization, and personalized experiences.

II. Background

A. Model Compression

The impressive achievements of deep neural networks (DNNs) in various domains come with the drawback of substantial computational complexity, memory footprints, and energy consumption. These factors pose significant obstacles to deployment on resource-constrained devices or in applications demanding real-time inference. Model compression has emerged as a vital area of research to address these limitations and to reduce the size and computational demands of DNNs while minimizing accuracy loss. Essential techniques include pruning redundant elements, quantizing weights and activations, applying low-rank factorization, distilling knowledge into smaller networks, and designing inherently efficient architectures. Model compression offers benefits like reduced storage needs, faster inference, improved energy efficiency, and expanded deployment possibilities. However, researchers

must navigate the compression-accuracy trade-off, address hardware compatibility issues, and often employ quantization-aware training [6]. Ongoing research in model compression focuses on automated compression strategies, fine-grained compression techniques, and hardware-aware optimization, paving the way for increasingly widespread and efficient use of DNNs.

B. Processing in-Memory

With their separation of processing and memory, traditional von Neumann computing architectures create a significant performance bottleneck due to the constant need for data movement. This "memory wall" becomes even more restrictive as datasets and AI workloads increase complexity. Processing-in-memory (PIM) presents a radical solution by integrating processing capabilities directly into or near memory units [7]–[9]. By minimizing data movement, PIM promises significant reductions in energy consumption and execution time while enabling massively parallel operations. Critical approaches to PIM include Processing Near Memory (PNM), which places processors close to memory, and Processing Using Memory (PUM), which leverages memory devices' properties for computation. While PIM holds the potential to transform data-intensive computing, challenges remain in developing suitable programming models, efficient data management techniques, system integration, and managing thermal considerations. With ongoing research in memory technologies and system design, PIM's benefits for applications like machine learning acceleration and real-time analytics make it a promising avenue for future computing paradigms.

III. DECO Architecture

A. Training Approaches

1) Network Architecture Design: In this work, we design a refined neural network inference framework tailored to the variable constraints of dynamic computing ecosystems. At the heart of this framework lies a judicious blend of a shared feature extractor and an array of task-specific modules anchored by model compression techniques to enhance adaptability and efficiency. This hybrid configuration is crafted from leading network architectures, selectively pared down, and compressed to yield a spectrum of leaner, differentiated models. Each model manifests a unique blend of computational complexity and task performance. By integrating a shared feature extractor, we eliminate redundant weight loading each time a new task module is selected, significantly conserving energy and enhancing efficiency in data movement across multiple neural network deployments. The inherent modularity of our design, complemented by model compression, is indispensable in edge computing scenarios, where it ensures an optimal trade-off between precision and computational thrift in edge computing, where the operational conditions can change rapidly.

2) Network Learning: In this segment, we meticulously outline the training methodologies designed to improve the performance of the variably configured networks introduced in Section III-A1.

Independent Learning: To establish a baseline, each task model undergoes independent training with a separate feature extractor for each classification module. This approach results in disparate parameters for the common feature extractor across task modules, necessitating frequent parameter reloading each time a new task module is selected. Thus, it leads to energy-inefficient data movement during inference. The learning objective for an image classification task using this approach is expressed in Equation 1.

$$L_{TM_i} = -\sum_{j=1}^{m} y_j \log \hat{y}_j \tag{1}$$

where TM_i represents a i^{th} task module and m denotes the distinct class categories defined for the task. Furthermore, y_j and $\hat{y}_j$ denote the ground-truth label and predicted label for a sample in j^{th} class, respectively.

Joint Learning: In contrast to training each task module separately, this approach involves training entire modules together with a common feature extractor. This ensures not only static weights for the common feature extractor, thus eradicating energy consumption for data movement, but it also facilitates mutual learning among modules. Consequently, the performance of each task module surpasses that achieved through independent training, serving as an improvement over the baseline method. The training objective for this approach, expressed in Equation 2, encompasses a linear combination of task loss across the different task modules, optimizing the network to achieve balanced performance that complies with the diverse requirements of each task.

$$L_t = -\sum_{i=1}^{n} L_{TM_i} \tag{2}$$

In the context of this training framework, n denotes the total count of task modules undergoing joint training, while L_{TM_i} signifies the individual training objective for i^{th} task module as expressed in Equation 1.

B. Digital In-Memory Accelerator

1) Overview: Our proposed PIM accelerator architecture, adopted from our preliminary work [10], is illustrated in Fig. 1(a), featuring computational sub-arrays, kernel and image banks, and a Digital Processing Unit (DPU) comprising three sub-components: Quantizer, Activation Function, and Batch Normalization. Control (Ctrl), situated within each sub-array, governs the execution of DNN layers.

The kernels (W) and input feature maps (I) are initially stored in Kernel and Image Banks, respectively, to facilitate mapping into sub-arrays. In phase (1), the operands are dispatched to the sub-arrays, tailored to manage the computational workload through the PIM mechanism. Subsequently, in phases (2) and (3), as elaborated further, the parallel computational sub-arrays execute feature extraction in conjunction with add-on peripherals, i.e., counter and shifter units. Ultimately, the accelerator's DPU activates the resulting feature map, producing the output feature map.

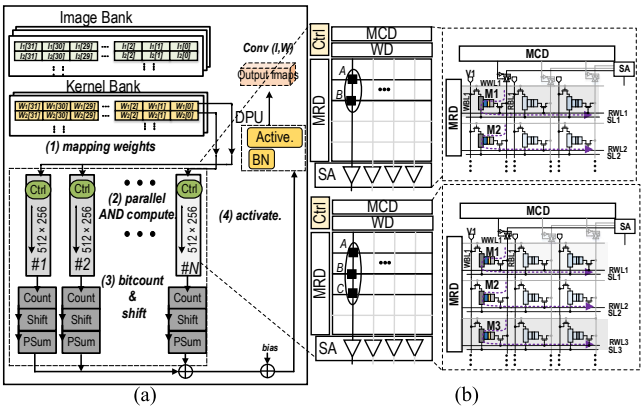

Fig. 1: (a) MRAM-based accelerator platform, (b) Computational sub-array design.

2) Sub-array architecture.: Fig. 1(b) depicts the in-memory computing sub-array architecture implemented utilizing binary SOT-MRAM cells [10], [11]. This sub-array comprises a Memory Row Decoder (MRD), a Memory Column Decoder (MCD), a Write Driver (WD), and Sense Amplifiers (SA), which are adjustable by Ctrl to enable dual-mode computation, encompassing memory write/read operations and in-memory logic operations. Each SOT-MRAM bit-cell within the computational sub-arrays is linked to five control signals, namely Write Word Line (WWL), Write Bit Line (WBL), Read Word Line (RWL), Read Bit Line (RBL), and Source Line (SL). The computational sub-array is primarily engineered to facilitate computation between in-memory operands utilizing two distinct mechanisms termed two-row activation and three-column activation. These mechanisms enable the implementation of bulk bitwise in-memory AND and addition operations, respectively.

3) Bit-line computation mode.: In the 2-/3-input in-memory logic method, pairs/triplets of bits stored in the identical column are typically selected with the MRD [12] and simultaneously sensed by SA connected to the corresponding bit-line. In this work, we adopted the reconfigurable SA from our previous work [10], depicted in Fig. 2. It comprises three

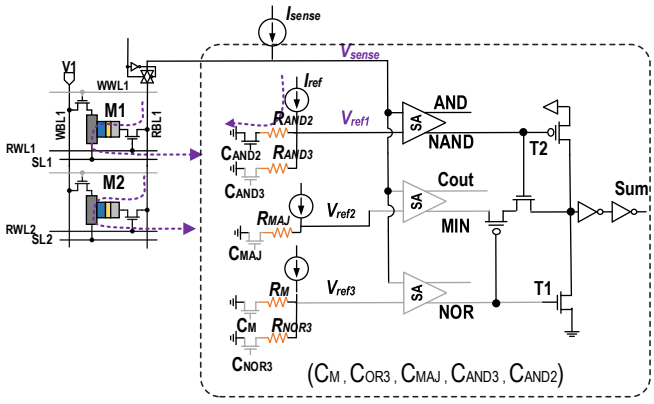

Fig. 2: The reconfigurable SA [10].

sub-SAs and a total of five reference-resistance branches that can be enabled by control bits (C_M, C_{OR3}, C_{MAJ}, C_{AND3}, C_{AND2}) under the control of the Ctrl unit, facilitating memory and computation modes. The SA can execute memory read and single-threshold logic operations by activating one control bit in the respective branches simultaneously. Furthermore, by activating multiple control bits simultaneously, SA can perform more than one logic function concurrently with the SAs. This capability can be leveraged to generate complex multi-threshold logic functions such as XOR3/XNOR3. The voltage generated after injecting a small reference current over the selected reference resistor by Ctrl is compared with the sensed voltage of equivalent resistance of the parallel-connected bit cells and their cascaded access transistors after injecting a small sense current over the resistors. By selecting different reference resistances, such as R_M and R_{AND2}, SA can perform primary memory and 2-input in-memory Boolean AND functions, respectively.

IV. Analysis and Evaluation

A. Experimental Setup

1) Dataset: Our methodology was thoroughly evaluated on two well-known datasets in image recognition: CIFAR-10 and CIFAR-100 [5]. CIFAR-10, a staple in computer vision, consists of 60,000 color images (32×32 pixels) divided into ten classes. CIFAR-100 expands on this with the same number and size of images but distributes them across 100 more detailed classes, increasing the complexity and diversity of the dataset.

2) Network Architectures: Central to our innovative edge inference framework is implementing versatile network architectures underpinned by a shared feature extractor and an array of specialized task modules. This architecture draws upon the proven strengths of seminal deep neural network models, specifically VGG16 [13] and MobileNetV1 [14], acclaimed for their robust edge inference capabilities in image classification domains. The initial convolutional layers of these renowned models form a consistent foundation for feature extraction, utilized across all classification modules without variation. Then, four distinct variants of classification modules for each network are crafted. The first classification module (CM), CM1, coupled with the common feature extractor (FE), mirrors the entire structure of the original models. The subsequent modules — CM2, CM3, and CM4 — are more streamlined versions, varying in size, computational demand, and energy requirements to suit different operational needs. The specific characteristics of each classification module, providing a granular view of their scale and complexity, are detailed in Table I.

3) Implementation Details: In our implementation, we employed the PyTorch framework for network creation, training, and testing, capitalizing on its versatility and efficiency. As part of our data preprocessing, we performed random horizontal flips as data augmentation and resized each input sample to $3 \times 64 \times 64$ to standardize the input dimensions. Optimization was achieved through stochastic gradient descent (SGD). We initiated training with a learning rate of 0.01

TABLE I: Detailed specification and performance evaluation of designed network modules

DNN Model	Modules	Convolutional Layers			Fully-Connected Layers			Total		Inference	Module
		# Layers	# MAC	# Params	# Layers	# MAC	# Params	# MAC	# Params	Time (ms)	Size (MB)
VGG16	FE	2	158,072,832	38,720	0	0	0	158,072,832	38,720	8.136	0.156
	CM1	11	1,094,713,344	14,675,968	3	3,155,968	3,158,026	1,097,869,312	17,833,994	1.150	71.712
	CM2	2	94,371,840	369,024	2	66,176	66,250	94,438,016	435,274	0.319	1.202
	CM3	1	75,497,472	73,856	2	66,816	66954	75,564,288	140,810	0.227	0.611
	CM4	0	0	0	2	17,024	17,098	17,024	17,098	0.139	0.093
MobileNetV1	FE	1	884,736	928	0	0	0	884,736	928	7.842	0.005
	CM1	26	1,572,864,000	3,206,048	1	10,240	10,250	1,572,874,240	3,216,298	2.241	13.361
	CM2	10	126,877,696	713,248	1	10,240	10,250	126,887,936	723,498	1.296	3.302
	CM3	8	134,217,728	29,728	1	1,280	1,290	134,219,008	31,018	0.797	0.184
	CM4	4	92,274,688	11,680	1	1,280	1,290	92,275,968	12,970	0.107	0.539

TABLE II: Performance comparison of the two training approaches across CIFAR-10 and CIFAR-100 datasets.

DNN Model	Experiment	CIFAR-10				CIFAR-100			
		CM1	CM2	CM3	CM4	CM1	CM2	CM3	CM4
VGG16	Independent Learning	89.38	85.09	81.92	75.74	63.57	55.1	53.36	45.31
	Joint Learning	90.59	85.07	82.76	76.04	65.44	53.82	53.15	45.94
MobileNetV1	Independent Learning	88.39	85.44	81.67	72.85	64.47	60.64	54.59	42.75
	Joint Learning	89.26	86.67	80.57	73.58	64.33	60.26	55.09	45.57

for each classification model, progressively reducing it by 90% after 80 epochs to enhance convergence. Training was conducted with a batch size of 128 over 200 epochs to ensure effective learning. Based on the validation set performance, the best-checkpoint was then evaluated on the test set to assess model generalization and performance concisely.

On the hardware side, we have configured our in-memory accelerator with a total capacity of 512Mb and a memory sub-array organized in a 256×512 layout using H-tree routing. For device simulations, we employed Non-Equilibrium Green's Function (NEGF) and Landau-Lifshitz-Gilbert (LLG) equations with spin Hall effect to model the SOT-MRAM bit-cell [11], [15], [16]. At the circuit level, we created a Verilog-A model for the 2T1R bit-cell, which can be utilized alongside interface CMOS circuits in Cadence Spectre. The performance metrics were evaluated using the 45nm NCSU PDK library [17]. At the architectural level, leveraging insights from device-circuit results, we extensively modified NVSim [18]. This modified simulator enables the adjustment of configuration files (.cfg) according to the model size and various memory array organizations. Subsequently, we developed a behavioral simulator in MATLAB to evaluate the energy and latency parameters associated with the accelerator's operation in running our PyTorch implementations.

B. Accuracy

A summary of the evaluation results of the various classification modules using the training frameworks defined in Section III-A2 across the two test sets is presented in Table II. It is discernible that the adoption of the joint learning paradigm engenders notable enhancements in model proficiency, particularly for streamlined classification modules, with some exceptions. The joint learning paradigm exhibited an average accuracy improvement of 0.7% and 0.52% for the VGG16 and MobileNetV1 architectures, respectively, on CIFAR-10. On the more complex CIFAR-100 dataset, these improvements were even more pronounced, with averages of 0.41% for VGG16 and 1.66% for MobileNetV1. These enhancements highlight the joint learning framework's ability to leverage synergies between shared feature extractors and classification modules, leading to more precise and energy-efficient neural network configurations.

C. Performance Evaluation

We assessed and reported the execution time and power consumption of different modules across two DNN models under examination for both GPU and PIM implementations in Table III. Regarding execution time, as anticipated PIM would outperform GPU implementation by a significant margin, demonstrating up to a two-order-of-magnitude increase in speed based on our experiment. For instance, with the CM1 module on MobileNet V1 with 1,572,874,240 multiply-accumulate (MAC) and 3,216,298 total number of parameters, PIM achieves a reduction in execution time by approximately 314-fold compared to GPU. This is primarily attributed to the highly parallelized PIM sub-array, which markedly accelerates MAC operations compared to the GPU. Furthermore, we note that as the number of MACs increases, greater parallelism and PIM sub-array utilization is achieved, resulting in reduced inference time. With regards to the power consumption of the

TABLE III: Power consumption and execution time comparison on CIFAR-100.

DNN Model	Module	Execution time (μs)		Power (W)	
		GPU	PIM	GPU	PIM
VGG16	**FE**	8130	24.2	350×2	1.433
	CM1	1150	11.6	350×2	0.211
	CM2	319	4.6	350×2	0.068
	CM3	227	3.8	350×2	0.059
	CM4	139	2.9	350×2	0.036
MobileNetV1	**FE**	7840	23.1	350×2	1.314
	CM1	13360	42.6	350×2	2.11
	CM2	1296	14.7	350×2	0.236
	CM3	797	9.1	350×2	0.132
	CM4	107	2.6	350×2	0.036

GPU, we derived a figure of 350W from the datasheets of the NVIDIA GeForce RTX 3090, which was then doubled to account for the use of two GPUs. Note that in this experimental setup, we have not excluded the power consumption associated with cooling systems and regulators. Overall, our observations indicate a significant reduction in power consumption with PIM implementation compared to GPU. Across a range of modules, PIM demonstrates a remarkable decrease in power consumption of up to four orders of magnitude.

V. Conclusion

DECO effectively combines model compression and PIM to improve deep learning efficiency on IoT devices. The shared feature extractor and joint learning leverage synergies and reduce redundancy, while the PIM accelerator minimizes data movement and exploits parallelism, leading to significant speedup and energy reduction over GPUs. Evaluation on CIFAR-10/100 using VGG16 and MobileNetV1 validates DECO's effectiveness. Its modular design suits dynamic edge environments, enabling intelligent edge analysis for responsive, privacy-preserving IoT applications.

Acknowledgement

This work is supported in part by the National Science Foundation under Grant No. 2216772, 2216773, 2303114, and 2247156.

References

[1] Y. Cheng *et al.*, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[2] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[3] R. Gaire *et al.*, "Encode: Enhancing compressed deep learning models through feature—distillation and informative sample selection," in *2023 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2023, pp. 633–638.

[4] A. Roohi *et al.*, "Enabling edge computing using emerging memory technologies: From device to architecture," in *Frontiers of Quality Electronic Design (QED) AI, IoT and Hardware Security*. Springer, 2022, pp. 415–464.

[5] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," 2009.

[6] M. Nagel *et al.*, "A white paper on neural network quantization. arxiv 2021," *arXiv preprint arXiv:2106.08295*.

[7] A. Roohi *et al.*, "Processing-in-memory acceleration of convolutional neural networks for energy-effciency, and power-intermittency resilience," in *20th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2019, pp. 8–13.

[8] S. Tabrizchi *et al.*, "Scima: A generic single-cycle compute-in-memory acceleration scheme for matrix computations," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 576–580.

[9] A. Roohi *et al.*, "Apgan: Approximate gan for robust low energy learning from imprecise components," *IEEE Transactions on Computers*, vol. 69, no. 3, pp. 349–360, 2019.

[10] S. Angizi *et al.*, "Graphs: A graph processing accelerator leveraging sot-mram," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 378–383.

[11] X. Fong *et al.*, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 1–22, 2015.

[12] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*, 2016, pp. 1–6.

[13] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[14] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[15] A. Roohi *et al.*, "Voltage-based concatenatable full adder using spin hall effect switching," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 12, pp. 2134–2138, 2017.

[16] R. Zand *et al.*, "Fundamentals, modeling, and application of magnetic tunnel junctions," in *Nanoscale Devices*. CRC Press, 2018, pp. 337–368.

[17] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[18] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, pp. 994–1007, 2012, doi: 10.1109/TCAD.2012.2185930.