Compute-in-Memory Upside Down: A Learning Operator Co-Design Perspective for Scalability

Shamma Nasrin, Priyesh Shukla, Shruthi Jaisimha, and Amit Ranjan Trivedi
Department of Electrical and Computer Engineering, University of Illinois at Chicago

***Abstract*—This paper discusses the potential of model-hardware co-design to simplify the implementation complexity of compute-in-SRAM deep learning considerably. Although compute-in-SRAM has emerged as a promising approach to improve the energy efficiency of DNN processing, current implementations suffer due to complex and excessive mixed-signal peripherals, such as the need for parallel digital-to-analog converters (DACs) at each input port. Comparatively, our approach inherently obviates complex peripherals by co-designing learning operators to SRAM's operational constraints. For example, our co-designed implementation is DAC-free even for multibit precision DNN processing. Additionally, we also discuss the interaction of our compute-in-SRAM operator with Bayesian inference of DNNs. We show a synergistic interaction of Bayesian inference with our framework, where Bayesian methods allow achieving similar accuracy with much smaller network size. Although each iteration of sample-based Bayesian inference is computationally expensive, the cost is minimized by our compute-in-SRAM approach. Meanwhile, by reducing the network size, Bayesian methods reduce the footprint cost of compute-in-SRAM implementation, which is a crucial concern for the method. We characterize this interaction for deep learning-based pose (position and orientation) estimation for a drone.**

***Index Terms*—Deep neural networks; Compute-in-memory; Pose-estimation; Nanodrone.**

I. Introduction

Deep neural networks (DNNs) are proliferating to various novel application spaces where high predictive accuracy alone is not adequate but low power and real-time performance is also equally critical. Since a DNN typically requires thousands to millions of parameters to achieve higher predictive capacity, a key challenge for employing DNNs in low power/real-time application platforms is its excessively high workload. Furthermore, a typical digital computing platform is *von Neumann*, i.e., has separate units for storage and computing. Therefore, the foremost challenge for digital processing of DNNs is due to excessive bandwidth demand between storage and computing.

To overcome the fundamental challenges due to excessive memory-processor bandwidth demand, *non-von Neumann* compute-in-memory approaches are gaining attention. In compute-in-memory DNN processing, memory modules are redesigned to be used for both storages of DNN's weights and processing them against inputs. Therefore, by using the same physical structure for computing and storage, compute-in-memory processing of DNN obviates repeated weight transfer from storage units to processors, limiting constraints on memory-processor bandwidth. However, compute-in-memory processing of DNNs is non-trivial. The primary function of conventional memories is to store. Thus, the additional requirement to *in situ* perform DNN's operations complicates their design complexity, limiting memory density and degrading energy efficiency/speed of processing.

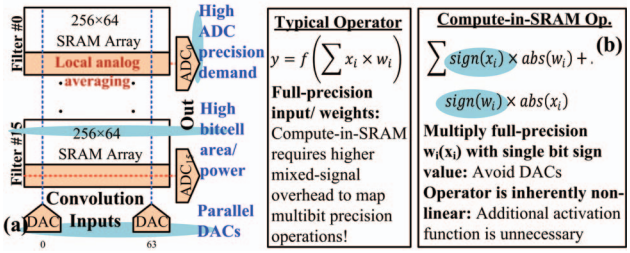


Figure 1: (a) Typical DNN operator requires computations between higher precision operands. Therefore, to implement them using compute-in-memory parallel digital-to-analog converters (DAC) and higher precision analog-to-digital converter (ADC) becomes necessary. (b) The proposed operator is compliant to compute-in-SRAM and obviates DACs even for multi-bit precision DNN while reduces precision demand on ADCs.

Due to the high potential of compute-in-memory for DNN and associated challenges, the field has become a rich playground for research from various disciplines. From the technology side, new non-volatile memory devices such as memristors [1]–[3], PCRAM [4]–[6], insulator-metal transition devices [7]–[9], spintronics [10], [11], Gaussian transistors [12]–[14], and even photonic devices [15]–[17] have been explored to simplify compute-in-memory DNN processing. From the architecture side, new schemes to overcome the rigidity of memory structures in the flexible mapping of DNNs have been proposed [18]–[20]. From the algorithm side, training methods for extremely low precision DNN have been explored [21]–[23].

While the above schemes have overcome the challenges of compute-in-memory to a varying degree, as the application scope of DNN continues to spread and power/performance-challenges aggravate, it has now become essential to explore a synergistic integration of multiple schemes to extract an even higher-order performance advantage. In this work, we specifically focus on the co-design of model and hardware for compute-in-memory to illustrate the potential of such *co-designed* approaches. As a test-case for our discussion, we consider the compute-in-SRAM implementation of DNN. Nonetheless, the design principles generalize beyond the test-case and are equally applicable to other memory structures.

II. Overview of Current-art of Compute-in-SRAM

The growing complexity of real-world applications necessitates DNNs operating with utmost energy efficiency and speed. Compute-in-SRAM has become a leading solution to enhance

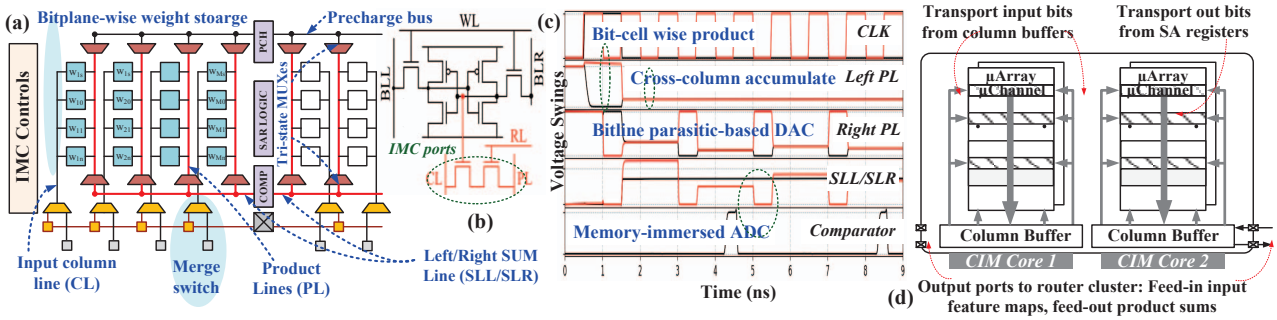


Figure 2: (a) Compute-in-SRAM based on the proposed operator. (b) Compute-in-SRAM cell. Additional ports in the cell (RL and PL) are utilized for *in situ* scalar product of input and weights. (c) Simulation waveforms. (d) Higher-level integration of compute-in-SRAM macros (μArrays) in a CIM-core.

the energy efficiency of DNNs due to two main reasons. First, the scheme minimizes data movement during DNN operations by collapsing computing and storage within the same physical structure. Secondly, the underlying memory technology, i.e., SRAM, is a high-speed structure designed on mass producible CMOS technologies, allowing easier on-chip integration with other system components. In [24], a six-transistor SRAM cell was used for binary-weighted neural networks. SRAM array was augmented with digital to analog converters (DACs) at each input port for mixed-signal within SRAM scalar product of inputs and 1-bit weights stored on SRAM cells. In [25], a charge mode within SRAM product operation was discussed, which also improved reliability against transistor-level process variability. In [26], time-domain DACs were used to minimize overheads of mixed-signal circuits. However, with increasing input precision, either operating time increases exponentially, or complex analog-domain voltage scaling is necessitated. In [27], DACs are precluded, but the operation is only limited to binary inputs and weights, which has low algorithmic accuracy. Multibit processing using DIMA architecture on 6-T SRAM was shown in [28] but requires on-chip training to overcome process variability. The scheme creates challenges at large scale production since each chip must be tested independently. In Supported BinaryNet [29], support parameters are applied through DAC to improve the prediction capacity of SRAM-mapped DNNs. In MC^{2}RAM [30], with-SRAM Markov chain Monte Carlo (MCMC) was shown for Bayesian deep learning operations. C3SRAM [31] utilizes mixed-signal capacitive coupling to evaluate a binary neural network; however, it incorporates one ADC per column, excessively increasing area/power overhead. In [32], a ReRAM based multibit, non-volatile CIM is proposed for Edge AI, operating at 2/4-bits input, 4-bit word line, and 10-bit output. A CIM based CNN processor is proposed in [33] with dynamic sparsity performance scaling architecture and exploits inter- and intra-CIM macro data reusability for energy efficiency. In [34], the fully parallel MAC CIM system's image recognition accuracy highly depends on ADC resolution, costing energy and latency.

III. A Learning Operator for Compute-in-SRAM

In our prior work [35], we discussed a novel neural network operator that considerably reduces the implementation complexity of compute-in-SRAM. In the new operator, the correlation of weight $\mathbf{w}$ and input $\mathbf{x}$ is represented as

$$\mathbf{w} \oplus \mathbf{x} = \sum_i \text{sign}(x_i) \cdot \text{abs}(w_i) + \text{sign}(w_i) \cdot \text{abs}(x_i) \quad (1)$$

Here, $\cdot$ is an element-wise multiplication operator, $+$ is element-wise addition operator, and $\sum$ is vector sum operator. sign() operator is ± 1 and abs() operator produces absolute unsigned value of the operand. Therefore, in Eq. 1, the correlation operator multiplies one-bit sign($\mathbf{x}$) against higher precision abs($\mathbf{w}$), and one-bit sign($\mathbf{w}$) against higher precision abs($\mathbf{x}$). Such decoupling of higher precision operands directly benefits compute-in-SRAM as discussed in the following.

In Figure 1(a), consider the compute-in-SRAM implementation of typical DNN operator using CONV-SRAM [25]. [25] uses a 10T bit-cell-based SRAM array to store the 1-b filter weights and a DAC for each column to generate analog inputs for the array to perform the dot product. An analog-to-digital converter is used to compute the partial digital outputs. Since DACs are concurrently active, they lead to high area and power overhead with the increasing precision of operands.

Comparatively, consider compute-in-SRAM using the proposed operator in Figure 1(b). To even simplify the implementation of Eq. 1, we consider the below reformulation:

$$\sum_i \text{sign}(w_i) \cdot \text{abs}(x_i) = 2 \sum_i \text{step}(w_i) \cdot \text{abs}(x_i) - \sum_i \text{abs}(x_i) \quad (2a)$$

$$\sum_i \text{sign}(x_i) \cdot \text{abs}(w_i) = 2 \sum_i \text{step}(x_i) \cdot \text{abs}(w_i) - \sum_i \text{abs}(w_i) \quad (2b)$$

In the above reformulation, $\text{step}(\cdot) \in [0, 1]$. $\sum_i \text{step}(w_i) \cdot \text{abs}(x_i)$ and $\sum_i \text{step}(x_i) \cdot \text{abs}(w_i)$ terms can be computed through compute-in-SRAM, as discussed below. $\sum_i \text{abs}(x_i)$ can be computed through a dummy row of weights, all storing ones. For a given input, this computation can be referenced for all weight vectors. $\sum_i \text{abs}(w_i)$ is a weight statistics that can be pre-computed and can be looked-up during evaluation.

If we consider a comparable compute-in-SRAM implementation for the proposed operator to Figure 1(a), DACs in the implementation can be obviated since the co-designed operator doesn't require operation between high precision operands. Note that in the proposed operator only 1-bit vectors sign($\mathbf{x}$) and sign($\mathbf{w}$) are operated against higher precision vectors abs($\mathbf{x}$) and abs($\mathbf{w}$), respectively. Meanwhile, to process l compute-in-SRAM cells in parallel, the typical operator requires l DACs, which is a considerable overhead and constrains its parallelism. With the proposed operator, each

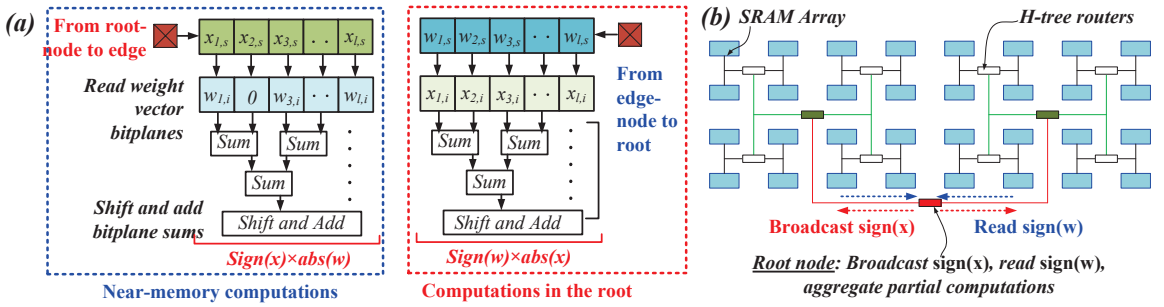


Figure 3: (a) Distributed processing using the proposed DNN operator. (b) H-tree-based spatial architecture combining multiple SRAM bank. Only sign($\mathbf{x}$) bit vector is transmitted from the root node to the distributed node and sign($\mathbf{w}$) bit vector is transmitted from the edge node to the root node.

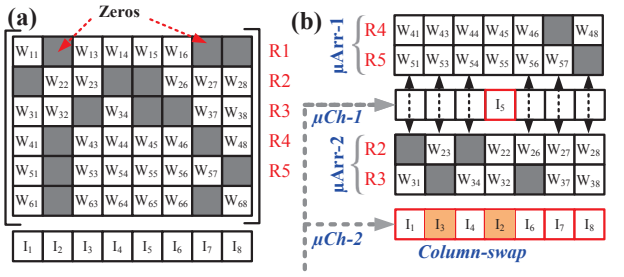
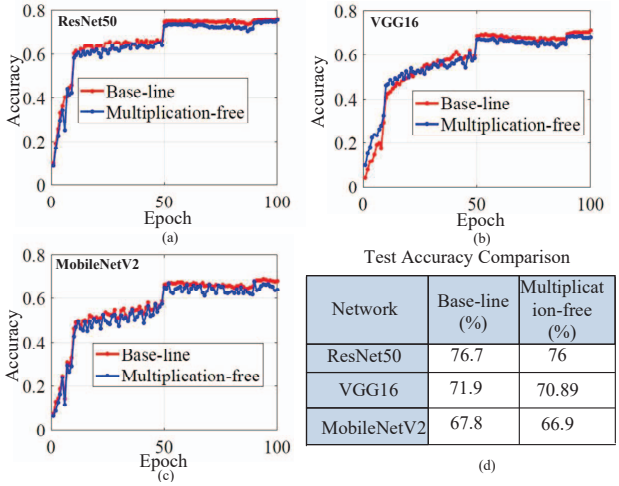


Figure 4: (a) Pruning can be exploited to avoid mapping zero weights to save space and power in compute-in-SRAM. (b) A fine-grained pruning scheme where weight rows with highest alignment of non-zero entries are grouped.



Test Accuracy Comparison

Network	Base-line (%)	Multiplication-free (%)
ResNet50	76.7	76
VGG16	71.9	70.89
MobileNetV2	67.8	66.9

(d)

Figure 5: Comparing the validation accuracy of multiplication-free operator based DNN versus the baseline for (a) ResNet50, (b) VGG16, and (c) MobileNetv2. (d) Test accuracy comparison across the three networks.

SRAM cell only performs a 1-bit logic operation; thus, to digitize the output of l columns, ADC with $log_2(l)$ precision is needed. Compare this to CONV-SRAM in Figure 1(a), where necessary ADC's precision is $n + log_2(l)$ since each SRAM cell processes n-bit DAC's output. By simplifying constraints on data converters, the co-designed operator also helps achieve higher vector-scale parallelism, i.e., processing a higher number of parallel columns (l) with the same ADC complexity in [25]. Additionally, the above step function reformulation of sign function will allow processing with a single product port of SRAM cells; thus, reducing dynamic energy. Comparatively, operations in CONV-SRAM are with weights $w \in [-1, 1]$, therefore, require differential ended processing and more complex cells.

IV. Detailed Design of Compute-in-SRAM using the Co-designed Operator

A. Compute-in-SRAM macro

Figure 2(a) shows the proposed design of compute-in-SRAM macro based on the co-designed operator. In the proposed design, an SRAM macro consists of μArrays and μChannels. Each μArray is dedicated to storing one weight channel. DNN weights are arranged across columns in a μArray, whereas each bit plane of weights is arranged in a row. Therefore, an N-dimensional weight channel with m-bit precision weights will require m rows and N columns of SRAM cells in a μArray. Figure 2(b) shows the proposed 8T SRAM cell used for the operator's in-SRAM processing. Extra transistors in the cell compared to a 6T cell decouple typical read/write operations to within cell product. The added transistors are selected by the row and column select lines (RL and CL) and operate on the product bit line (PL). The decoupling of read/write and product operations mitigates interference between the operations, reduces the impact of process variability, and allows operation in storage hold mode. μChannels convey digital inputs/outputs to/from μArrays. If a weighting filter has many channels, μChannels also allow stitching of μArrays so that inputs can be shared among the μArrays. Moreover, the same SRAM array can also be used for memory-immersed *in situ* analog-to-digital conversion (ADC) as discussed in our prior work [35]. Figure 2(c) shows the exemplary waveforms for within-SRAM inner product computation and digitization. Figure 2(d) shows higher-level integration of compute-in-SRAM macros. A compute-in-memory (CIM) core comprises parallel processing columns. Within a column, μArrays are stacked vertically. μChannels interleave μArrays and transport inputs to them from the input feature-map buffer at the bottom. An output channel from CIM-columns transports product-sum bits to the output buffer at the bottom.

B. Near-memory adaptation of co-designed operator

The co-designed operator is not only suited for compute-in-memory but also for near-memory processing. Figure 3 shows an example implementation of H-tree-based spatial architecture for near-memory processing using the operator. In the ensuing discussion, we highlight how the proposed operator can naturally minimize data communication between

distributed nodes. To compute $\mathbf{w} \oplus \mathbf{x}$, *only* the bit sign vectors of $\mathbf{w}$ and $\mathbf{x}$ travel between root nodes and edge nodes. Root node pushes $\text{sign}(\mathbf{x})$ vector to SRAM banks, and SRAM arrays only communicate $\text{sign}(\mathbf{w})$. At the end of processing, root node collects partial product-sums and combines them to determine DNN output. Therefore, communication overheads do not grow proportional to the operating precision. To compute $\sum \text{sign}(\mathbf{x}) \cdot \text{abs}(\mathbf{w})$ near an SRAM array, weight vector bitplanes $\mathbf{w}_j$ are sequentially read. By controlling column sense-amplifiers with the logic values of $\text{sign}(\mathbf{x})$, only those weights where the respective $\text{sign}(\mathbf{x})$ is one are read. An adder tree is used to sum the bit of element-wise product vector. A similar adder tree and processing are used to evaluate $\sum \text{sign}(\mathbf{w}) \cdot \text{abs}(\mathbf{x})$ in the root node.

C. Interaction of novel operator with pruning approaches

Minimal weights in a DNN can be ignored without much impact on the prediction accuracy. If a weight value is zero, it need not be stored and processed. However, the rigid physical structure and processing flow of compute-in-SRAM makes it challenging to exploit weight sparsity. Interestingly, our compute-in-SRAM learning operator ($\oplus$) can also help to address this challenge. We earlier discussed that $\oplus$ obviates DACs in DNN processing. This led to the design of low overhead DAC-free μChannels. Meanwhile, fine-grained interleaving of μChannels in a CIM-column [Figure 2(d)] can provide it the flexibility to efficiently map sparse weight matrices. With current CIM approaches relying on typical operator and DACs, fine-grained interleaving of peripherals is impractical due to excessive overheads. For example, Sparse ReRAM [36] operates with fine-grained weight matrices to exploit sparsity, like us, but can only process them sequentially to limit peripheral overheads. Therefore, efficiency in handling sparsity comes at the cost of low throughput. Figure 4(b) shows an example mapping strategy where μChannels help μArrays exploit sparsity. Using procedures such as [37], [38], a sparse matrix can be partitioned into sub-matrices where rows in the sub-matrix have aligned sparsity, thereby allowing elimination of common zero columns. The reduced sub-matrices can be mapped together in μArrays. A CIM-core can combine columns of varying dimensions so that after zero-column elimination, the reduced matrix can be placed on a column of matching width. The alignment of two sub-matrices can be computed by counting the number of sharing input indices. Sub-matrices with the highest alignment can be placed on μArrays in proximity. Therefore, pruned networks can be effectively implemented by merging between two μArrays. If two columns are merged, inputs are passed to the top array directly from the bottom array, and the loading of input bits is bypassed on the top column; therefore, overheads to load input feature-map can be minimized.

D. Accuracy comparison to the typical operator

We characterize our proposed DNN operator's prediction accuracy on the benchmark ResNet50, VGG16, and MobileNetV2 networks. These networks are typically used as a backbone for many computer vision tasks. As shown in Figures 5(a)-(c), the validation accuracy of the proposed multiplication-free operator based DNNs is comparable to the baseline networks. In Figure 5(d), we compare the test accuracy between the baseline DNNs and multiplication-free operator based DNNs. The multiplication-free operator-based network's test accuracy is competitive to that based on a typical operator and drops less than one percent. With a little compromise in accuracy, the multiplication-free operator significantly reduces the implementation complexity compared to the typical.

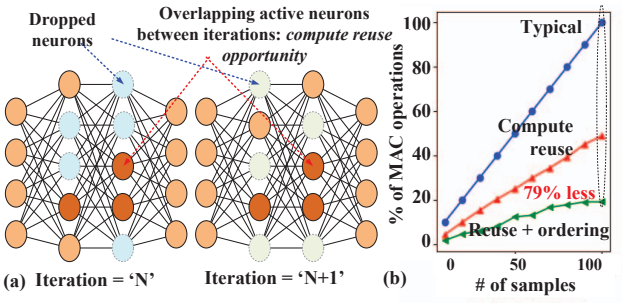


Figure 6: (a) Dropout iterations and compute reuse opportunities. (b) Necessary MAC operations at varying number of dropout samples by considering compute reuse and optimal sample ordering.

V. Synergy if Compute-in-SRAM with Bayesian Inference

Even though the traditional (i.e., deterministic) DNNs achieve remarkable accuracy in many complex decision-making tasks, flawless predictions can not be guaranteed. To address this challenge, Bayesian inference (BI) of DNN is gaining attention. Unlike deterministic inference, BI can rigorously account for the model and training uncertainties. Predictions in BI can express the prediction confidence. For example, BI-based predictions might say 'Greenlight with 70% confidence,' unlike just saying 'Greenlight' as in deterministic inference. If the underlying DNN is not confident, top-level controllers can take risk-aware safeguarding.

The predictive robustness of BI comes with an overwhelming computing cost, nonetheless. The excessive workload of BI has stimulated research on alternate frameworks such as variational inference (VI) [39]. A novel VI framework, Monte Carlo Dropout (MC-Dropout) [40], was recently proposed where dropout was exploited for VI during the test. In the method, dropout used during training is also implemented during the test. Predictions from multiple iterations are combined to capture the prediction's statistical moments, such as the mode/mean determines the output, and the variance determines prediction confidence. In the following discussion, we show the potential of a synergistic integration of Compute-in-SRAM with such Bayesian inference methods.

A. Compute reuse in sampling domain

In Figure 6(a), consider an MC-Dropout layer where input and output neurons are dropped randomly in each iteration to estimate the partial sum over several iterations. However, two successive iterations can share a common set of active input/output neurons (highlighted in the figure), thus presenting an opportunity to reuse computations. This can also be seen in Figure 6(b), where iteratively computing the product sum in N+1th iteration from Nth iteration will only require eval-

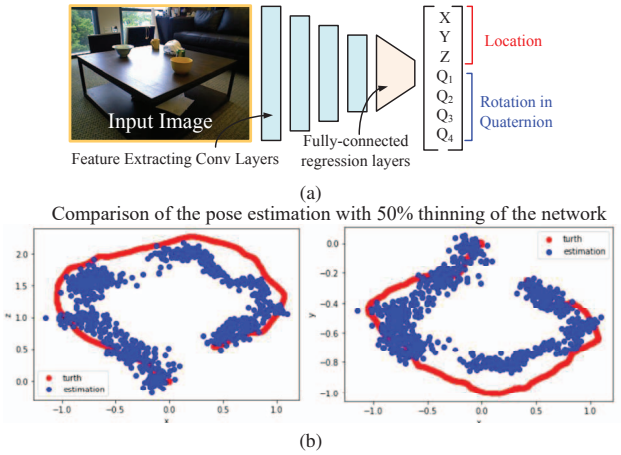


Figure 7: (a) Deep learning-based visual odometry: pose estimation based on RGB input frame stream. (b) Prose estimation of the RGB image for 50% thinning of the network.

uation of non-overlapping input and output neurons between the iterations (i.e., corresponding matrix columns and rows, highlighted in color in the figure). Our compute-in-SRAM framework can exploit this by operating only on SRAM columns that correspond to the non-overlapping input neurons. Working on fewer columns reduces dynamic energy as well as necessary ADC's precision. For near-memory implementation, the iterative compute flow minimizes the energy and latency by operating on fewer operands.

B. Optimal sample ordering to maximize compute reuse

Since each iteration's compute cost depends on the overlap of input/output neurons from the previous iteration, the compute reuse can become even more effective if dropout samples can be optimally ordered. Notably, the above compute reuse opportunities only depend on weight samples and not on inputs; thus, our framework can benefit from offline sampling and ordering. Interestingly, this sample ordering is equivalent to a traveling salesman's problem (TSP). In an analogy to TSP, samples here represent cities, and sample-to-sample compute cost means the distance between the cities. We can further enhance this reuse efficiency if we expand compute reuse to past n samples where partial sums from all these previous samples can be reused. The equivalent sample ordering problem, in this case, is the traveling repairman's problem (TRP). TSP and TRP are well-studied optimization problems. TRP-based sample ordering can be more effective but will require more registers to store intermediate product sums. In Figure 6(b), we compare the necessary MAC operations at varying numbers of samples by considering typical flow, compute reuse flow, and compute reuse with optimal sample ordering where the last saves computations by 79% for 100 samples.

C. Simulation Results

We characterize the interaction of our framework and Bayesian inference for the pose estimation of a drone in an indoor setting. As shown in Figure 7, the input RGB image captured by the drone is extracted by convolutional layers and finally regressed by fully-connected layers to output the predicted pose (i.e. x-y-z location and orientation expressed as

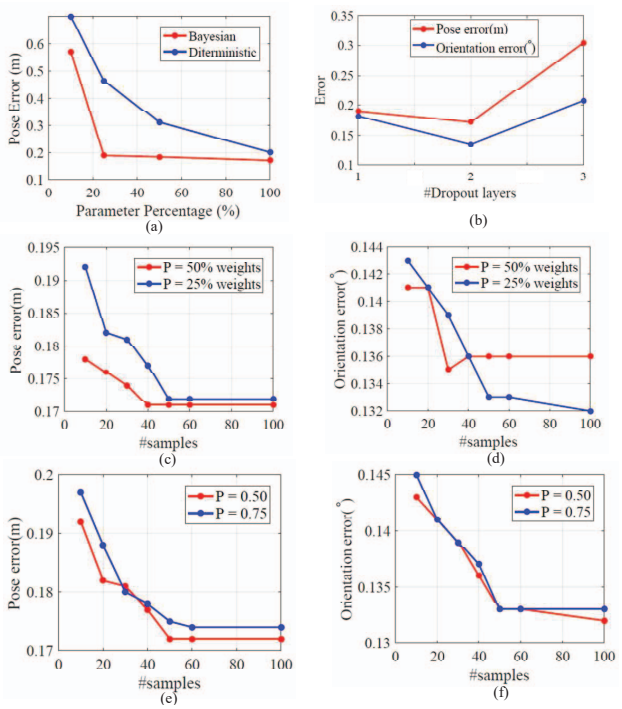


Figure 8: (a)-(f) Pose estimation and orientation error for Bayesian inference with different level of network thinning.

quaternions q1-q2-q3-q4 of the drone camera). The network is trained using RGB-D dataset [41] comprising of image frames, and the corresponding camera poses for various indoor settings. Since the predictive confidence of Bayesian inference (BI) comes with an overwhelming computing cost, thinning down the network reduces energy overhead for BI significantly. Using the RGB-D dataset, we trained and tested different network configurations, considering BI's energy constraints. In Figure 8, we discuss the relocalization error of RGB images at different levels of thinning of the network. We use Bayesian GoogleNet to regress the camera pose from a single RGB image. Figure 8(a) compares the position error with different thinning levels for the deterministic and Bayesian networks. In figure 8(b), we choose the optimum number of dropout layers to be added to the GoogleNet during training and inference. Figure 8(c)-(f) show that even a very thin network with only 25% of learnable parameters can restore the accuracy of the full network with an optimum number of Bayesian (dropout) network samples.

VI. Conclusions

Compute-in-Memory has emerged as a critical approach to energy-efficient deep learning. Although most of the current research focuses on low-level design innovations, such as more efficient cell design and processing flow, in this work, we have presented a new perspective on compute-in-memory where the learning operator itself is co-adapted to memory's physical and processing constraints. Using such co-designing, in-memory deep learning doesn't require DAC, reduces ADC's precision requirement, and is readily scalable to multibit precision operations. Our proposed operator based compute-in-SRAM achieves competitive accuracy to the typical operator

based DNNs. We have also shown the interaction of Bayesian inference (BI) methods with compute-in-memory where BI methods can reduce the network size and compute-in-memory can reduce the sample iteration cost of BI. To further enhance BI for compute-in-memory, we have presented a novel sample ordering approach to maximize compute reuse.

Acknowledgement: We thankfully acknowledge the support from Intel's gift funding for this work.

REFERENCES

[1] R. Tetzlaff, *Memristors and memristive systems*. Springer, 2013.

[2] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[3] A. Thomas, "Memristor-based neural networks," *Journal of Physics D: Applied Physics*, vol. 46, no. 9, p. 093001, 2013.

[4] S. T. Harshfield and D. Q. Wright, "Pcram memory cell and method of making same," Sep. 5 2006, uS Patent 7,102,150.

[5] K. et al, "Nvm neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning," in *IEEE International Electron Devices Meeting (IEDM)*, 2015, pp. 17–1.

[6] S. Ambrogio, P. Narayanan, H. Tsai, R. M. Shelby, I. Boybat, C. di Nolfo, S. Sidler, M. Giordano, M. Bodini, N. C. Farinha *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, pp. 60–67, 2018.

[7] M. Taha, S. Walia, T. Ahmed, D. Headland, W. Withayachumnankul, S. Sriram, and M. Bhaskaran, "Insulator–metal transition in substrate-independent vo 2 thin film for phase-change devices," *Scientific reports*, vol. 7, no. 1, pp. 1–10, 2017.

[8] A. Parihar, M. Jerry, S. Datta, and A. Raychowdhury, "Stochastic imt neurons: An interplay of thermal and threshold noise at bifurcation," *Frontiers in neuroscience*, vol. 12, p. 210, 2018.

[9] M. Belyaev and A. Velichko, "A spiking neural network based on the model of vo2–neuron," *Electronics*, vol. 8, no. 10, p. 1065, 2019.

[10] S. Nasrin, J. Drobitch, P. Shukla, T. Tulabandhula, S. Bandyopadhyay, and A. R. Trivedi, "Bayesian reasoning machine on a magneto-tunneling junction network," *Nanotechnology*, vol. 31, no. 48, p. 484001, 2020.

[11] S. Nasrin, J. L. Drobitch, S. Bandyopadhyay, and A. R. Trivedi, "Low power restricted boltzmann machine using mixed-mode magneto-tunneling junctions," *IEEE Electron Device Letters*, vol. 40, no. 2, pp. 345–348, 2019.

[12] A. R. Trivedi and A. Shylendra, "Ultralow power acoustic feature-scoring using gaussian IV transistors," *Design Automation Conference (DAC)*, pp. 1–6, 2018.

[13] M. E. Beck, A. Shylendra, V. K. Sangwan, S. Guo, W. A. G. Rojas, H. Yoo, H. Bergeron, K. Su, A. R. Trivedi, and M. C. Hersam, "Spiking neurons from tunable gaussian heterojunction transistors," *Nature communications*, vol. 11, no. 1, pp. 1–8, 2020.

[14] A. Sebastian, A. Pannone, S. S. Radhakrishnan, and S. Das, "Gaussian synapses for probabilistic neural networks," *Nature communications*, vol. 10, no. 1, pp. 1–11, 2019.

[15] I. Chakraborty, G. Saha, and K. Roy, "Photonic in-memory computing primitive for spiking neural networks using phase-change materials," *Physical Review Applied*, vol. 11, no. 1, p. 014063, 2019.

[16] C. Ríos, N. Youngblood, Z. Cheng, M. Le Gallo, W. H. Pernice, C. D. Wright, A. Sebastian, and H. Bhaskaran, "In-memory computing on a photonic platform," *Science advances*, vol. 5, no. 2, p. eaau5759, 2019.

[17] Y. Vlasov, "Silicon photonics for next generation computing systems," in *European Conference on Optical Communications*, 2008, pp. 1–2.

[18] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocessors and Microsystems*, vol. 67, pp. 28–41, 2019.

[19] B. Kim, S. Lee, A. Trivedi, and W. J. Song, "Energy-efficient acceleration of deep neural networks on realtime-constrained embedded edge devices," *IEEE Access*, 2020.

[20] N. Iliev, A. Gianelli, and A. R. Trivedi, "Low power speaker identification by integrated clustering and gaussian mixture model scoring," *IEEE Embedded Systems Letters*, vol. 12, no. 1, pp. 9–12, 2019.

[21] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[22] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[23] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. Cheung, and G. A. Constantinides, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–39, 2019.

[24] R. Liu, X. Peng, X. Sun, W.-S. Khwa, X. Si, J.-J. Chen, J.-F. Li, M.-F. Chang, and S. Yu, "Parallelizing sram arrays with customized bit-cell for binary neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[25] A. Biswas and A. P. Chandrakasan, "Conv-sram: An energy-efficient sram with in-memory dot-product computation for low-power convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 217–230, 2018.

[26] M. Kang, S. Lim, S. Gonugondla, and N. R. Shanbhag, "An in-memory vlsi architecture for convolutional neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 494–505, 2018.

[27] Z. Jiang, S. Yin, J.-s. Seo, and M. Seok, "Xnor-sram: In-bitcell computing sram macro based on resistive computing mechanism," *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pp. 417–422, 2019.

[28] M. Kang, S. Gonugondla, and N. R. Shanbhag, "A variation-tolerant dima via on-chip training." Springer, 2020, pp. 81–100.

[29] S. Nasrin, S. Ramakrishna, T. Tulabandhula, and A. R. Trivedi, "Supported-binarynet: Bitcell array-based weight supports for dynamic accuracy-energy trade-offs in sram-based binarized neural network," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020.

[30] P. Shukla, A. Shylendra, T. Tulabandhula, and A. R. Trivedi, "Mc2ram: Markov chain monte carlo sampling in sram for fast bayesian inference," in *IEEE International Symposium on Circuits and Systems*, 2020.

[31] Z. Jiang, S. Yin, J.-s. Seo, and M. Seok, "C3sram: An in-memory-computing sram macro based on robust capacitive coupling computing mechanism," *IEEE Journal of Solid-State Circuits*, 2020.

[32] C.-X. Xue, T.-Y. Huang, J.-S. Liu, T.-W. Chang, H.-Y. Kao, J.-H. Wang, T.-W. Liu, S.-Y. Wei, S.-P. Huang, W.-C. Wei *et al.*, "15.4 a 22nm 2mb reram compute-in-memory macro with 121-28tops/w for multibit mac computing for tiny ai edge devices," *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 244–246, 2020.

[33] J. Yue, Z. Yuan, X. Feng, Y. He, Z. Zhang, X. Si, R. Liu, M.-F. Chang, X. Li, H. Yang *et al.*, "14.3 a 65nm computing-in-memory-based cnn processor with 2.9-to-35.8 tops/w system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse," *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 234–236, 2020.

[34] Q. Liu, B. Gao, P. Yao, D. Wu, J. Chen, Y. Pang, W. Zhang, Y. Liao, C.-X. Xue, W.-H. Chen *et al.*, "33.2 a fully integrated analog reram based 78.4 tops/w compute-in-memory chip with fully parallel mac computing," *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 500–502, 2020.

[35] S. Nasrin, D. Badawi, A. E. Cetin, W. Gomes, and A. R. Trivedi, "Mf-net: Compute-in-memory sram for multibit precision inference using memory-immersed data conversion and multiplication-free operators," *arXiv preprint*, 2020.

[36] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," *International Symposium on Computer Architecture*, pp. 236–249, 2019.

[37] A. Yzelman and R. H. Bisseling, "Cache-oblivious sparse matrix–vector multiplication by using sparse matrix partitioning methods," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 3128–3154, 2009.

[38] Ü. i. t. V. Çatalyürek, C. Aykanat, and B. Uçar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 656–683, 2010.

[39] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

[40] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *international conference on machine learning*, pp. 1050–1059, 2016.

[41] K. Lai, L. Bo, and D. Fox, "Unsupervised feature learning for 3d scene labeling," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3050–3057.