frontiers

LRMP: Layer Replication with Mixed Precision for Spatial In-memory DNN Accelerators *

Abinand Nallathambi [1,⋆], **Christin David Bose** [1], **Wilfried Haensch** [2] **and Anand Raghunathan** [1]

[1] *Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, USA*
[2] *Materials Science Division, Argonne National Laboratory, Argonne, Illinois, USA*

Correspondence⋆:
Abinand Nallathambi
anallath@purdue.edu

ABSTRACT

In-memory computing (IMC) with non-volatile memories (NVMs) has emerged as a promising approach to address the rapidly growing computational demands of Deep Neural Networks (DNNs). Mapping DNN layers spatially onto NVM-based IMC accelerators achieves high degrees of parallelism. However, two challenges that arise in this approach are the highly non-uniform distribution of layer processing times and high area requirements. We propose LRMP, a method to jointly apply layer replication and mixed precision quantization to improve the performance of DNNs when mapped to area-constrained NVM-based IMC accelerators. LRMP uses a combination of reinforcement learning and mixed integer linear programming to search the replication-quantization design space using a model that is closely informed by the target hardware architecture. Across five DNN benchmarks, LRMP achieves 2.8-9$\times$ latency and 11.8-19$\times$ throughput improvement at minimal ($< 1\%$) degradation in accuracy.

Keywords: RRAM, in-memory computing, analog accelerator, quantization, reinforcement learning, mixed integer linear programming

1 INTRODUCTION

Deep Neural Networks (DNNs) have come to dominate the field of machine learning, and achieve state-of-the-art performance on a variety of complex tasks. However, the advancements in their capabilities have come at the cost of a steep growth in model sizes and computational complexity. Researchers have developed specialized digital accelerator architectures (Jouppi et al. (2018); Lie (2022); Chen et al. (2016)) and methodologies like quantization and pruning (Liang et al. (2021)) that attempt to strike better tradeoffs between cost and functional performance. These implementations are, however, fundamentally limited by the memory bottleneck, as memory accesses are significantly more expensive than digital compute.

In-Memory Computing (IMC) is a computing paradigm where the elementary operations of input vector-weight matrix multiplications in DNNs are performed within memory arrays, potentially alleviating the memory bottleneck. IMC systems have been designed and prototyped with various memory technologies, including SRAM (Kang et al. (2020); Yin et al. (2020); Zhang et al. (2017)), DRAM (Gao et al. (2019)),

*This work was supported in part by the U.S. Department of Energy, Office of Science, for support of microelectronics research, under contract number DE-AC0206CH11357, and in part by the National Science Foundation under grant CCF-2106964

and emerging non-volatile memories (NVM) such as RRAM (Shafiee et al. (2016); Chi et al. (2016); Song et al. (2017)), PCM (Burr et al. (2015); Narayanan et al. (2021); Khaddam-Aljameh et al. (2021)) and STT-MRAM (Yan et al. (2018); Jain et al. (2018)). In this work, we focus on IMC with emerging NVMs, where the weights are programmed into the NVM arrays as the resistance, or conductance, of the memory device. To perform a vector-matrix multiplication (VMM) using such a memory array, the input vector can be presented simultaneously along multiple wordlines using digital-to-analog converters (DACs). The current that flows through each memory cell is the product of the resistance of the memory element (weight) and the wordline voltage (input). These currents naturally sum up at each bit line. These behaviors are dictated by Ohm's and Kirchoff's laws. These bit line currents can then be digitized using analog to digital converters (ADCs) to produce the input vector-weight matrix dot product.

Emerging non-volatile memories have a lot of desirable qualities that make them a good fit for in-memory computing. Their high density means that larger models can be stored, and their non-volatility eliminates the need for continuous power or refresh. However, their high programming costs coupled with their limited endurance make frequent weight re-programming undesirable. These factors make NVMs suitable for weight-stationary inference architectures where all the weights are programmed spatially across the chip and activations flow through and get processed by the appropriate arrays. A consequence of this approach is that the required area scales with the size of the network. While NVM arrays are compact, the peripherals required for IMC (ADCs and DACs) can be quite large, lowering the effective density (storage capacity per unit area) and thus, resulting in large area requirements. To mitigate the large area requirements, researchers have proposed bit-decomposed architectures (Shafiee et al. (2016); Ankit et al. (2019)), in which weight bits are stored in spatially distinct arrays and input bits are processed serially, reducing the precision requirements of ADCs and DACs. As the area of the peripherals scale with their precision, the reduced precisions of ADCs and DACs in bit-decomposed architectures result in better effective density, thereby lowering area requirements of IMC accelerators.

While mitigating the area requirements is important, the system performance is also an important consideration. Despite the impressive peak performance offered by spatial IMC accelerators, the actual performance achieved can be significantly lower due to poor utilization caused by the non-uniformity in processing times across the layers of a DNN. Effective mapping techniques can help close the gap between peak and actual performance (Jain et al. (2023)). *Layer replication*, which replicates bottleneck layers to facilitate tensor and data parallelism, can balance layer processing times and thus, improve performance. However, layer replication is not a trivial optimization. Finding the resources to replicate the layers, choosing the right layers to replicate, and the number of times to replicate the chosen layers, are all non-trivial decisions that involve complex tradeoffs between area requirements and performance. Also, with growing model sizes, improving the performance of spatial architectures with layer replication becomes challenging since it exacerbates the area requirements.

Researchers have proposed various techniques to address these varied challenges of IMC accelerator design. For example, various mixed precision quantization techniques that assign specialized bitwidths to the weights and activations across the layers of a DNN using different optimization strategies (Peng et al. (2022); Meng et al. (2021b); Kang et al. (2021); Huang et al. (2021)) have been shown to achieve significant compression of weight and activation footprints, and resulting in area, performance and efficiency improvements. However, these techniques do not address the severe under-utlization of NVM tiles caused by the imbalance in processing times across the layers of DNNs. Others have proposed layer replication techniques to improve utilization that either do not address the question of where to find the resources to replicate layers (Rasch et al. (2019); Li et al. (2020, 2023a)) or rely on design-time

tradeoffs to accommodate the replicated layers (He et al. (2022)). In contrast, we identify a novel synergy between **L**ayer **R**eplication and **M**ixed **P**recision quantization that can be exploited at compilation-time to improve the performance of DNNs on spatial IMC accelerators. We propose LRMP, an automated mapping framework that explores the quantization-replication design space to quantize layers selectively to free up resources and then replicate the right layers using the freed-up resources to improve performance. In summary, our contributions are as follows:

- We present LRMP, a novel framework that jointly performs mixed precision quantization and layer replication during mapping of DNNs to improve system performance on IMC accelerators.
- We propose a joint-optimization approach with a deep reinforcement learning based framework that selects the precision for each layer in the network with regard to performance and accuracy to conserve hardware resources, and a linear programming based technique to selectively replicate layers by reutilizing the conserved resources in the IMC hardware to improve performance.
- We evaluate the LRMP framework on a benchmark suite of convolutional and fully-connected neural networks and achieve $2.8 - 9\times$ latency improvement and $11.8 - 19\times$ throughput improvement, at iso-area and near iso-accuracy.

The rest of the paper is organized as follows. We describe the process of mapping a neural network layer in an IMC system and discuss the implications of precision on resource requirements and latency in Section 2. We motivate the synergy between mixed precision and layer replication using an illustrated example in Section 3. We present the details of the LRMP framework in Section 4. We describe our experimental setup in Section 5 and present our results in Section 6. We discuss the contributions of our work in the context of existing related works in Section 7. Finally, we conclude the paper in Section 8.

2 PRELIMINARIES

Vector-matrix multiplication (VMM) is an elementary operation in the evaluation of neural networks. In this section, we describe how a weight matrix can be mapped to multiple crossbar tiles and how these crossbar tiles can collectively perform a multiplication operation between an input vector and a weight matrix to produce an output vector. We also describe the latency and the number of crossbar tiles required for such an implementation of VMM.

Convolutional layers represent a common layer configuration used in DNNs. They are composed of a three-dimensional weight tensor array sliding across a three-dimensional input tensor, producing an output value for each patch of overlap. Convolutional layers are realized on IMC substrates by converting the weight tensor into a two-dimensional matrix and performing image-to-column lowering of the input tensor into a sequence of vectors. Then, the convolution output values can be produced by performing a sequence of VMMs.

Consider a convolution operation with C input features, N output features and a kernel size of K, producing an output features of dimension $W \times W$. The size of its lowered weight matrix is $K^2C \times N$. The input tensor is transformed into W^2 vectors of length K^2C. With a sequence of VMMs, W^2 output vectors of length N are produced. It must be noted that the number of vectors can be quite high and it depends on the dimensions of the input, the filter kernel size K, the padding and the stride. For, example, in the first convolutional layer of the ResNet18 DNN, the input matrix has over 12,000 column vectors.

To map a convolutional layer to a crossbar, the weight tensor is first lowered to a two-dimensional matrix, as shown in Figure 1. The weight matrix is then segmented into multiple sub-matrices of size $X \times X$, which

denotes the size of the crossbar array or tile. To build a spatial architecture, each of these sub-matrices are mapped onto individual crossbar tiles. The number of tiles required to perform this spatial mapping is given by Equation 1.

$$\#\text{tiles}(K, C, N, X) = \left\lceil \frac{K^2 C}{X} \right\rceil \times \left\lceil \frac{N}{X} \right\rceil \tag{1}$$

As discussed in Section 1, crossbar arrays can be built with a variety of memory technologies. Also, these memory devices are designed to store a specific number of bits. The precision of the memory element is a matter of concern, as high precision elements have been shown to be more sensitive to process variations and conductance drift (Shim et al. (2021)) and incur higher programming costs (Perez et al. (2021)). Thus, low precision devices are desirable with regards to both accuracy and performance.

The achievable precision of the memory device needs to be reconciled with the required logical precision of the weights. If the device precision (s_b) is less than the required weight precision (w_b), the weight sub-matrices can be sliced into groups of s_b bits and then each slice can be mapped to a separate crossbar tile, as shown in Figure 1. It must be noted that the digital outputs corresponding to the bit-slices of the weight matrix need to be appropriately shifted and added to produce the final output. The number of tiles required considering a bit-sliced mapping is given by Equation 2.

$$\#\text{tiles}(K, C, N, X, w_b, s_b) = \left\lceil \frac{K^2 C}{X} \right\rceil \times \left\lceil \frac{N}{X} \right\rceil \times \left\lceil \frac{w_b}{s_b} \right\rceil \tag{2}$$

As discussed in Section 1, to perform a vector-matrix multiplication using crossbars, we must convert the input vector values to analog currents using a digital-to-analog converter (DAC). The output of the

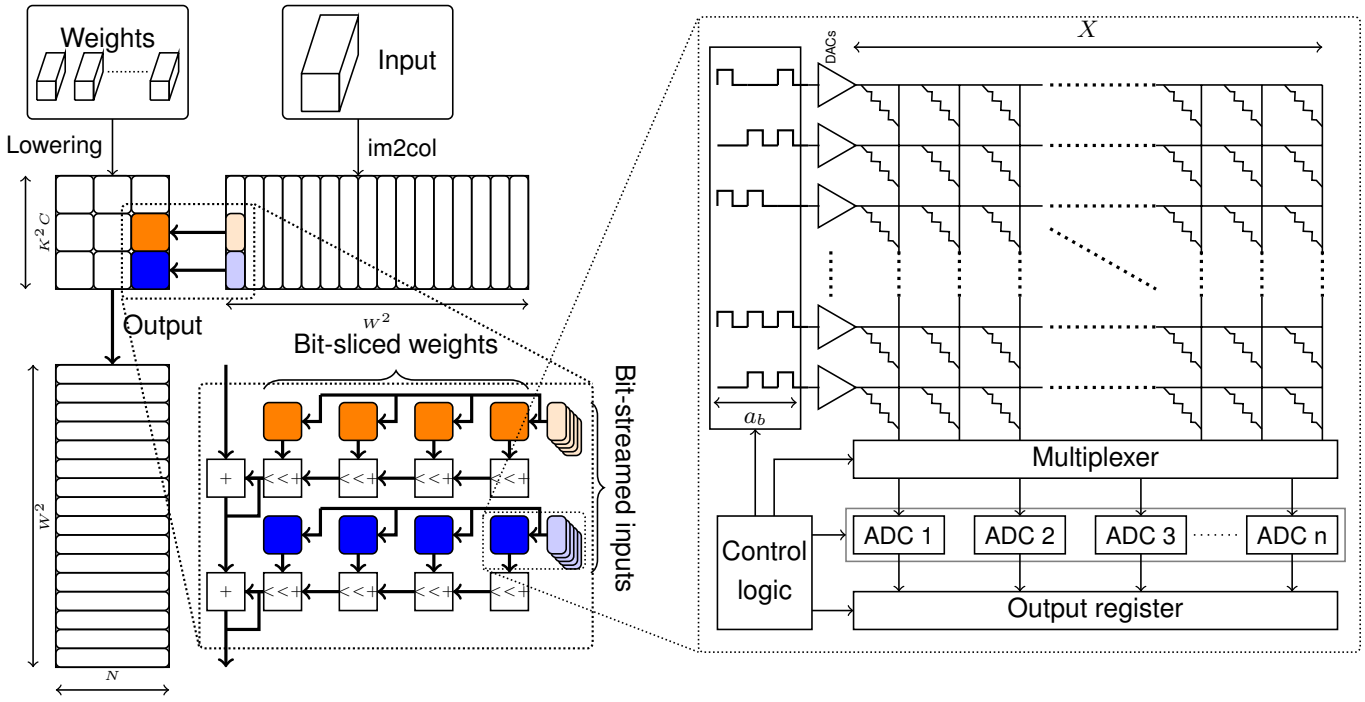

Figure 1. Functional description of a VMM operation using crossbar arrays

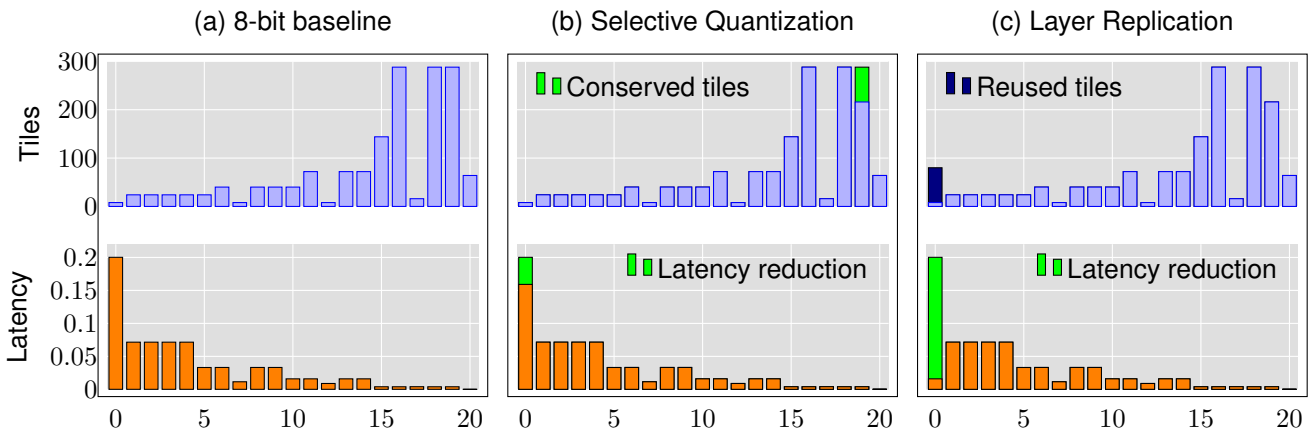

Figure 2. An experimental illustration of heterogeneous quantization and layer replication using ResNet18

crossbar array is then converted to digital using an analog-to-digital converter (ADC). These peripheral circuits, especially ADCs, occupy a significant proportion of latency and, area and power budgets. The design choices of the number of ADCs per crossbar array, and ADC and DAC precisions are important considerations in the design of crossbar-based architectures. The number of ADCs can be chosen to be lesser than the number of columns and the ADCs can be time-multiplexed between multiple columns. In order to reduce the precision requirements of ADCs and DACs, we can stream the input vectors bit-by-bit and reduce the corresponding outputs with shift-add operations. The latency of performing VMMs required by a convolution layer in such a bit-streamed manner is given by Equation 3.

$$\text{lat}(W, a_b, X, n_{ADC}, t_{tile}) = W^2 \times t_{tile} \times \left\lceil \frac{X}{n_{ADC}} \right\rceil \times a_b \tag{3}$$

where n_{ADC} is the number of ADCs per crossbar array, t_{tile} is the time elapsed between presenting an input to the tile and the ADCs producing their output, a_b is the number of bits required to represent the input vector values and W^2 is the number of vectors.

Thus, both the hardware requirements and latency of a crossbar-based architecture depend on the precision of the weights and activations of the neural networks mapped onto them.

3 MOTIVATION

In this section, we illustrate the impact of mixed precision on tile allocation and latency/throughput of evaluation. We also demonstrate the benefits of selectively replicating layers in a DNN.

Let us consider the baseline implementation of ResNet18 with 8-bit weights and 8-bit activations. As defined by Equation 2, the tile consumption of each layer in a spatial architecture depends on the size of the weight matrix and the precisions of weights (w_b) and memory device (s_b). As shown in Fig. 2(a), we observe that different layers of the network have different latencies and tile costs, as defined by Equations 3 and 2, respectively. In our evaluations, we use a device precision of 1-bit and a crossbar size of 256×256.

By selectively reducing the precision of weights in certain layers, we can take advantage of the bit-sliced implementation and reduce the number of tiles required by that layer. These conserved tiles can be used to replicate bottleneck layers in a neural network to process multiple input vectors in parallel, resulting in

a latency reduction. Similarly, by selectively reducing the precision of input vectors, we can reduce the number of bits to be streamed to the crossbar arrays, resulting in proportional reduction of latency.

Let us consider reducing the weight precision of a resource-intensive layer and the input precision of the bottleneck layer to 6-bits. As shown in Fig. 2(b), we observe that 72 tiles are conserved. In addition, as explained by Equation 3, the latency of the bottleneck layer is reduced resulting in an overall 5.7% latency improvement and 1.33× throughput improvement.

If these newly freed-up tiles are used to naively replicate only the bottleneck layer, we can create 9 more copies of that layer. Thus, 10 input vectors of that layer can be processed in parallel, resulting in 25.5% improvement in total latency and 2.34× improvement in throughput, as illustrated in Fig. 2(c).

The above example illustrates that a trade-off exists between precision and latency in spatial IMC architectures. A few considerations that arise about this trade-off are:

How to choose the precision of each layer?

When choosing the precision of each layer, we need to consider its impact on the tile consumption, overall latency, and accuracy. The weight precision affects the bit-slicing factor, which is only one of the factors that determines the tile consumption of a layer (Equation 2). The other factors depend on the size of the weight matrix. Thus, it is important to choose the weight precision of each layer in a way that the number of tiles conserved is maximized. Similarly, the activation precision only affects the bit-streaming factor of Equation 3. The other factor is the number of input vectors to be processed. Thus, it is important to choose the activation precision of each layer in a way that the latency is minimized. Moreover, reducing the activation/weight precision of any layer in a neural network has implications for the overall accuracy of the network. Thus, it is important to choose the precision of each layer in a way that the overall accuracy is not compromised.

Where to repurpose the conserved tiles?

When choosing the replication factor of a layer, we need to consider its impact on the overall latency and tile consumption. The latency of a layer is a function of the number of input vectors and their precision, both of which can vary across the layers of a neural network. At the same time, the number of tiles required to replicate layers also varies based on the size of their weight matrices. Thus, it is important to choose the replication factor of each layer in a way that the utility of the conserved tiles is maximized, and the overall latency is minimized.

4 LRMP METHODOLOGY

In this paper, we propose LRMP (Layer Replication through Mixed Precision), a framework that combines a reinforcement learning (RL) and mixed integer linear programming (MILP) to jointly optimize latency/throughput and accuracy of DNNs realized on IMC hardware fabrics.

As shown in Figure 3 LRMP is an iterative process with each iteration or episode consisting of two-steps: (1) an RL-agent choosing the precision of each layer in the DNN, and (2) an MILP-based optimizer choosing the replication factors of each layer. After each episode, the latency/throughput achieved by the MILP-based optimizer and the accuracy of the network are used to train the RL-agent. In the remainder of this section, we describe the hardware modeling of an RRAM-based IMC accelerator, and then discuss how linear programming and reinforcement learning can be employed in tandem to quantize and replicate layers to jointly optimize accuracy and performance metrics under a chip capacity constraint.

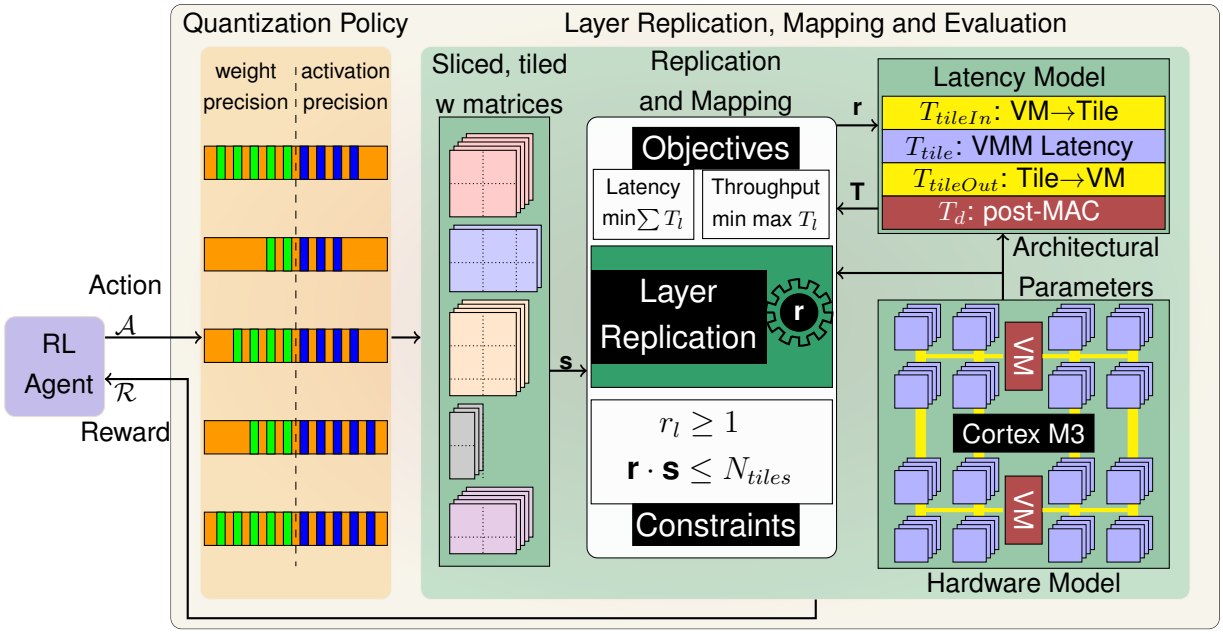

Figure 3. Overview of the proposed LRMP methodology.

4.1 Hardware Model

LRMP is a joint optimization process that is designed to improve accuracy and latency/throughput achieved by DNNs on spatial in-memory accelerators. To estimate the performance metrics of evaluating DNNs on these spatial accelerators, we develop a simple and effective cost model that can be used to estimate latency and throughput of a DNN on the fly. The cost model is based on the compute-in-memory system developed by Chang et al. (2022), which consists of a Cortex M3 microprocessor, two vector modules for digital compute, and 288 crossbar tiles of dimension 256×256. Data transport is implemented using 8 lanes of 8-bit wide buses from the vector modules to the crossbar tiles and 8 lanes of 32-bit wide buses from the crossbar tiles to the vector modules. Each vector module has 8 lanes of parallel compute and 128KB of SRAM. The system is equipped with fine-grained power gating and each tile can be individually turned off. On account of the larger computer vision models used to benchmark the proposed approaches in this work, the cost model assumes a scaled-up version of this system with 5682 tiles and 40 vector modules, each with 64 lanes of parallel compute. Further details on the microarchitecture are provided in Section 5.

The latency of evaluating a DNN layer (T_l) on this compute-in-memory system, described in Eqn. 4, comprises of 4 factors: $T_{tileIn.l}$, which refers to the latency of transferring input vectors of layer l from vector modules to the respective tiles. This latency is shared by eight 8-bit lanes, which are shared among 144 tiles. Similarly, $T_{tileOut.l}$ represents the latency of transferring output vectors of layer l from crossbar tiles to the respective vector modules using eight 32-bit lanes shared by the same 144 tiles. The latency of performing Vector-Matrix-Multiplication (VMM) using crossbar tiles with temporally bit-streamed inputs and spatially bit-sliced weights of layer l is represented by $T_{tile.l}$. Lastly, the post-VMM digital compute of layer l performed by vector modules has a latency of $T_{d.l}$, which uses 64 lanes processing the output vectors of 144 tiles. It must be noted that each of these components is a function of the number of bits used to represent the input activations and weights of layer l.

$$T_l = T_{tileIn.l} + T_{tileOut.l} + T_{tile.l} + T_{d.l} \tag{4}$$

With the latency of evaluating a layer defined, the latency of evaluating a DNN is the sum of the latencies of its constituent layers. The latency of evaluating a DNN with L layers is given by Eqn. 5.

$$T = \sum_{l=1}^{L} T_l \tag{5}$$

The system is designed to operate with coarse-grained pipeline parallelism. Thus, the throughput of the system is defined by the maximum latency of any layer. The throughput of the system is given by Eqn. 6.

$$P = \frac{1}{\max\limits_{l} T_l} \tag{6}$$

The model described above is used in LRMP to perform analysis and exploration of the quantization and replication design space.

4.2 Optimizing layer replication using mixed integer linear programming

As discussed in Section 3, tiles can be freed up by selectively quantizing layers based on their tile footprint and selectively replicating layers based on their latencies. When a layer is replicated, the number of tiles and vector modules allocated to that layer is increased. Thus, for the said layer: (1) the total bandwidth available for data transfer is increased; (2) the amount of digital compute allocated is increased; and (3) the number of tiles available for performing the required VMM operations is increased. This results in a linear reduction in the latency of evaluating the layer.

If there are r_l instances of layer l, then the latency of evaluating the DNN is given by Eqn. 7.

$$T = \sum_{l} \frac{1}{r_l}(T_{tileIn.l} + T_{tileOut.l} + T_{tile.l} + T_{d.l}) \tag{7}$$

Given a mixed precision quantization scheme, the total number of tiles required to have one instance of each layer (s_l) is given by Eqn. 2. The total latency T can be optimized by carefully choosing the layer replication factors $\mathbf{r}$. The process of choosing the replication factors is naturally constrained by the total number of tiles available in the system (N_{tiles}). This can be formulated as a constrained optimization problem, as shown in Formulation 8.

$$\begin{aligned} &\underset{\mathbf{r}}{\text{minimize}} \quad \sum_{l} \frac{1}{r_l}(T_{tileIn.l} + T_{tileOut.l} + T_{tile.l} + T_{d.l}) \\ &\text{subject to} \\ &r_l \geq 1, \\ &\sum_{l}(r_l * s_l) \leq N_{tiles} \end{aligned} \tag{8}$$

The constraints ensure that there is at least one instance of each layer and the total number of allocated tiles doesn't exceed the number of tiles available (N_{tiles}). Since s_l is constant for a given quantization scheme, the constraints are linear. However, the objective function is non-linear.

As defined by Eqn. 6, the throughput of the system is the inverse of the maximum latency across all layers. Thus, to maximize throughput, we need to minimize the maximum latency across all layers. Optimizing for throughput is, thus, a min-max problem. The optimization problem can therefore be re-written as shown in Formulation 9.

$$\begin{aligned} &\underset{\mathbf{r}}{\text{minimize}} \quad M \\ &\text{subject to} \\ &\frac{1}{r_l}(T_{tileIn.l} + T_{tileOut.l} + T_{tile.l} + T_{d.l}) \leq M, \\ &r_l \geq 1, \\ &\sum_l (r_l * s_l) \leq N_{tiles} \end{aligned} \tag{9}$$

We introduce a dummy variable M and reformulate the optimization problem to minimize M, while ensuring that the latency of each layer does not exceed M. By constraining the latency of each layer to be less than M and minimizing M, we are effectively minimizing the maximum latency across all layers, which maximizes the throughput of the system.

These optimization problems are not automatically linear. However, we can employ linearization techniques (Asghari et al. (2022)) to reformulate the optimization problem with linear constraints and objective function. Then, we solve the reformulated problem using an MILP solver.

4.3 Constraining the action space with performance budgets

The reinforcement learning framework used in this work is based on the work by Wang et al. (2019), which imposes a performance cost constraint on the action space of the RL agent. If the quantization policy prescribed by the RL agent does not meet the performance targets, it is modified by decreasing the bitwidths until the performance targets are met. While this approach is effective, it does not provide any insight into the tradeoffs between accuracy and performance. We restructure this approach to explore the tradeoffs between accuracy and performance by exponentially tightening the performance budget. This results in the RL agent exploring the space of quantization policies to not just meet a performance budget but also achieve better performance metrics.

4.4 Rewarding the RL agent with accuracy and performance metrics

In each episode of exploration, the RL agent is rewarded based on the quality of the quantization policy it prescribes. Wang et al. (2019) rewarded the RL agent based on the accuracy of the quantized DNN. In this work, we optimize the performance of the quantized DNN by using the layer replication technique. Thus, to achieve joint optimization, the RL agent is rewarded based on the accuracy and performance of the quantized DNN. The reward function is given by Eqn. 10.

$$\mathcal{R} = \lambda \times (acc_{quant} - acc_{original}) + \alpha \times (1 - T_{quant}/T_{original}) \tag{10}$$

where, acc_{quant} is the accuracy of the quantized DNN, $acc_{original}$ is the accuracy of the original DNN, T_{quant} is the latency of the quantized DNN and $T_{original}$ is the latency of the original DNN when optimizing for latency. When optimizing for throughput, T_{quant} and $T_{original}$ are latencies of the bottleneck layers

of the respective DNNs. The hyperparameters λ and α control the relative importance of accuracy and performance in the reward function. The reward function is designed to encourage the RL agent to prescribe quantization policies that result in a quantized DNN that is optimized to balance accuracy and speed.

5 EXPERIMENTAL METHODOLOGY

5.1 Microarchitectural details

As described in Section 4, this work is based on a scaled-up model of the compute-in-memory system fabricated by Chang et al. (2022). The microarchitectural parameters are listed in Table 1.

The system is built using 1T-1R RRAM eNVM technology, with a tile size of 256x256 and a total of 5682 tiles. The system also includes 40 vector modules, each of which contains 64 lanes of parallel digital compute and 128 KB of SRAM. Each tile is equipped with eight 4-bit Flash ADCs and 256 1-bit DACs. To prevent partial sum quantization and mitigate other non-idealities, only 9 rows are activated at a time. The system is clocked at 192 MHz.

The energy consumption of the system is modeled with three components: power consumed by the RRAM tiles, the energy cost of reading and writing the activations to the on-chip SRAM buffers, and the power leaked by the SRAMs. Each RRAM tile is reported to consume an average power of 70 μW (Chang et al. (2022)). The SRAM blocks are modelled using CACTI.

While we evaluate LRMP on a specific architecture, the proposed techniques are not specific to this architecture. The proposed optimizations are applicable to any bit-decomposed IMC architecture(Shafiee et al. (2016); Zhu et al. (2019); Ankit et al. (2019)), and are otherwise agnostic to the underlying hardware.

Parameter	Value
eNVM	1T-1R RRAM
Tile size	256×256
No. of tiles	5682
No. of vector modules	40
Device precision	1 bit
Row parallelism	9
DAC precision	1 bit
Column parallelism	8
ADC precision	4 bits
Avg. power per tile	70 μW
Clock frequency	192 MHz

Table 1. Microarchitectural parameters

5.2 Methods

Reinforcement learning

As described in Section 4, the reinforcement learning framework used in this work is based on the hardware-aware quantization tool proposed by Wang et al. (2019). The method consists of two phases: exploration and finetuning. In the exploration phase, the agent explores the action space to find a good

policy based on the performance budget and rewards provided, as described in Section 4. The trajectory of the exploration phase is discussed in Section 6.3.

After the exploration phase, the DNN is quantized with the mixed precision scheme found by the agent. In the finetuning phase, the DNN is trained with the quantized weights and activations to recover any accuracy lost to quantization.

Mixed Integer Linear programming

Given a quantization policy prescribed by the RL agent, the mixed integer linear programming step is used to find the replication factors that optimize the performance of the system. Optimization objectives of both latency (latencyOptim) and throughput (throughputOptim) are implemented. The baseline for each network in the benchmark suite is the implementation with 8-bit weights and activations. Thus, the layer replication is done with a constraint that the total number of tiles used is no more than the baseline. This is a design choice to ensure that performance is optimized while utilizing the same number of tiles as the baseline. An ablation study has been performed and described in Section 6.5 to show the effectiveness of our LRMP method with or without this design constraint.

5.3 Benchmarks

The proposed LRMP approach has been evaluated on a set of DNN benchmarks trained on the ImageNet and MNIST datasets. The baseline of comparison for each benchmark is the implementation with 8-bit weights and activations. The benchmarks are listed in Table 2, along with the number of tiles required by the baseline implementation. The multilayer perceptron (MLP) is trained on the MNIST dataset, with 4 hidden layers of 1024, 4096, 4096 and 1024 neurons respectively. The ResNets are finetuned on the ImageNet dataset with pre-trained weights. While we limit the evaluation of proposed techniques to DNNs trained for classification tasks, we believe the proposed techniques are broadly applicable to any quantized DNN(Dettmers et al. (2022), Li et al. (2023b)). Also, the proposed techniques do not place any limits on the number of bits used for quantization.

It must be noted that, besides quantization, analog non-idealities such as noise, conductance drift, device-to-device variation etc. have not been modeled in this work. However, modeling these non-idealities (Roy et al. (2021); Jain et al. (2020); Lu et al. (2021)) and developing compensation techniques (Meng et al. (2021a); Jain and Raghunathan (2019); Charan et al. (2020)) are areas of active and ongoing research and we believe these effects are not an impediment to the principal contributions of this work.

Benchmark	**Dataset**	N_{tiles}
MLP	MNIST	3232
ResNet18	ImageNet	1602
ResNet34	ImageNet	2965
ResNet50	ImageNet	3370
ResNet101	ImageNet	5682

Table 2. DNN benchmarks

6 RESULTS

In the sub-sections of this section, we first present the latency, throughput and energy improvements achieved by LRMP. We then present results that provide insights into the RL-based exploration process.

We also show a layer-wise breakdown of how latencies are optimized and an ablation study that analyses the sensitivity of the layer replication methodology to area constraints.

6.1 Latency and throughput improvements

Fig. 4 reports the latency and throughput improvements achieved by the LRMP framework. As explained in Section 5, the improvements are reported with respect to fixed-precision baseline networks with 8-bit weights and activations.

We observe 2.8-9× reduction in latency and 8-15× improvement in throughput while optimizing for latency (denoted as latencyOptim) across the suite of benchmark DNNs. Similarly, we observe 11.8-19× improvement in throughput and 2.5-8× reduction in latency while optimizing for throughput (denoted as throughputOptim). These improvements are obtained with accuracy loss of less than 1% after finetuning with the quantization policies determined by the exploration phase of LRMP.

6.2 Energy improvements

Although LRMP explicitly optimizes for throughput or latency, it achieves energy improvements as a result of more efficient DNN execution on the IMC substrate. Fig. 5 shows the energy improvements achieved by

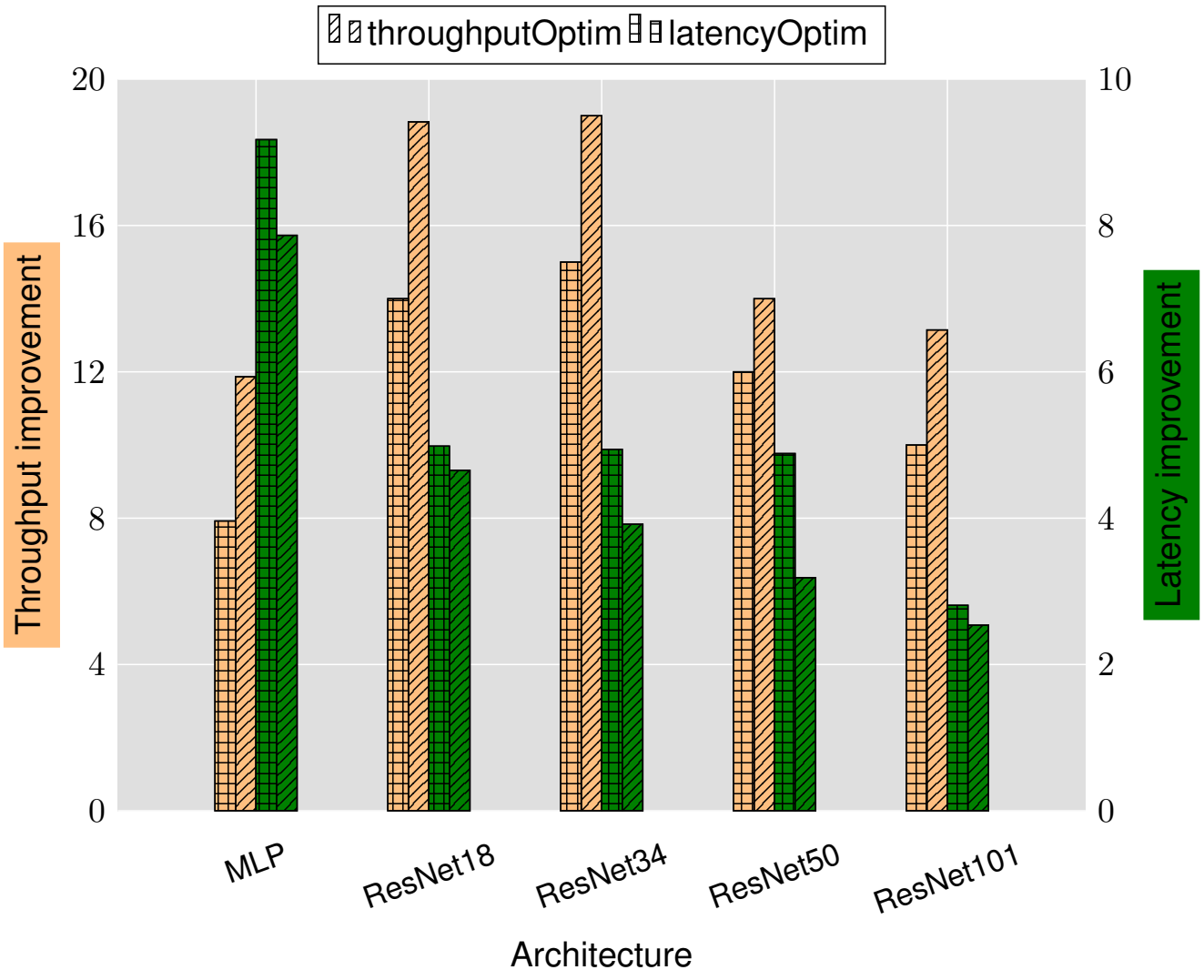

Figure 4. Latency and throughput improvements achieved by LRMP

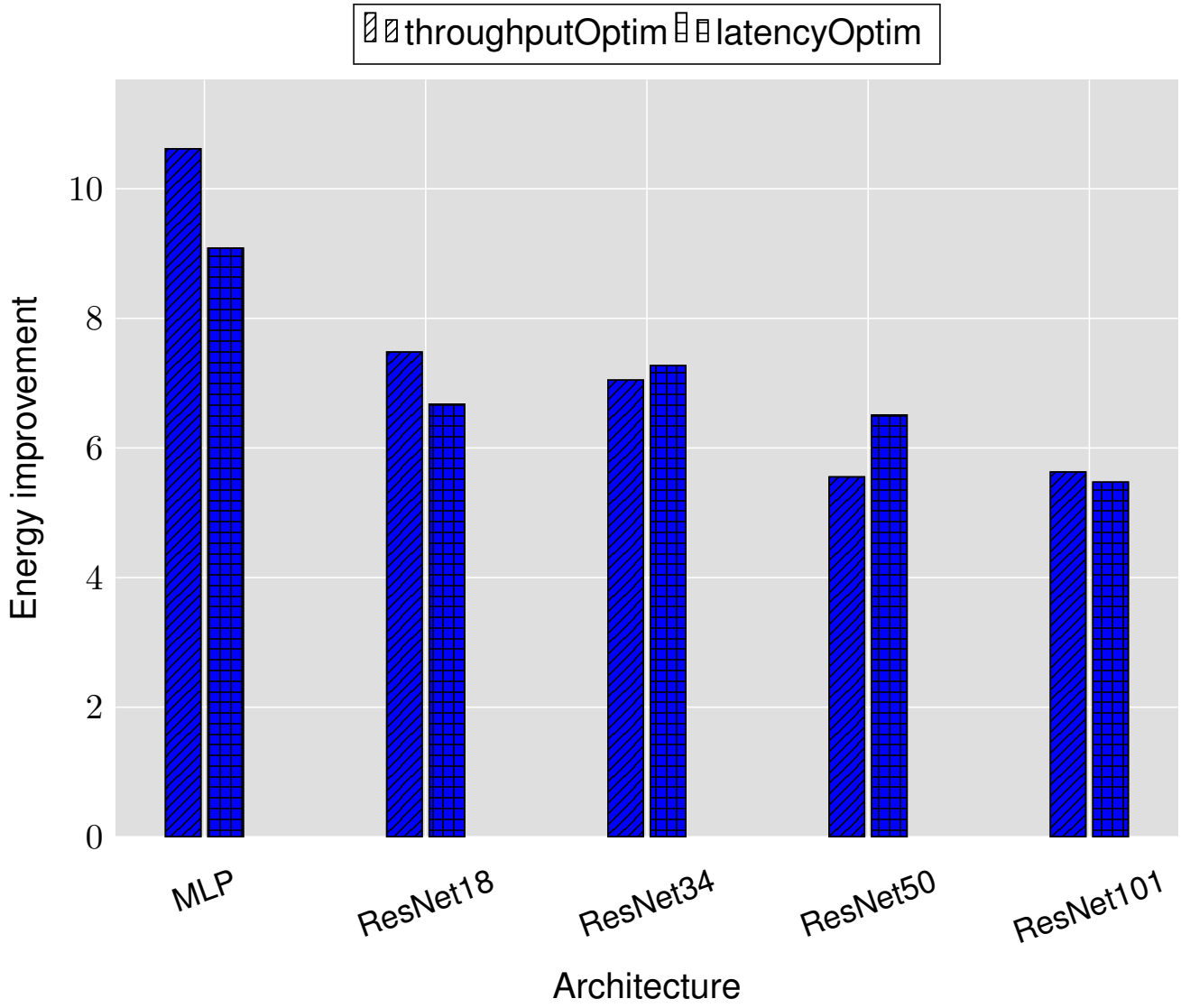

Figure 5. Energy improvements achieved by LRMP

LRMP. We observe 5.5-10.6× improvement in energy consumption while optimizing for throughput and 5.5-9× energy improvement while optimizing for latency.

6.3 Studying joint optimization of accuracy and performance

As discussed in Section 4, the proposed approach jointly optimizes for accuracy and performance by rewarding the RL agent with an affine combination of accuracy and performance metrics, and by continuously tightening the constraints placed on the action space. Fig. 6 shows the trajectory of the RL agent performing latency optimization for ResNet18. The exploration is started with a lenient performance budget of 0.35×baseline latency and exponentially tightened to 0.2×baseline latency. Over the course of the exploration, the agent finds quantization policies that achieve upto 5× improvement in latency with layer replication while also improving the accuracy.

6.4 Layer-wise breakdown

As discussed in Section 4, the layer replication can be performed by optimizing for either latency or throughput. The two objectives have different implications on the latencies and tile consumptions of each layer and thus, latency and throughput outcomes for the overall network. Fig. 7 shows the layer-wise

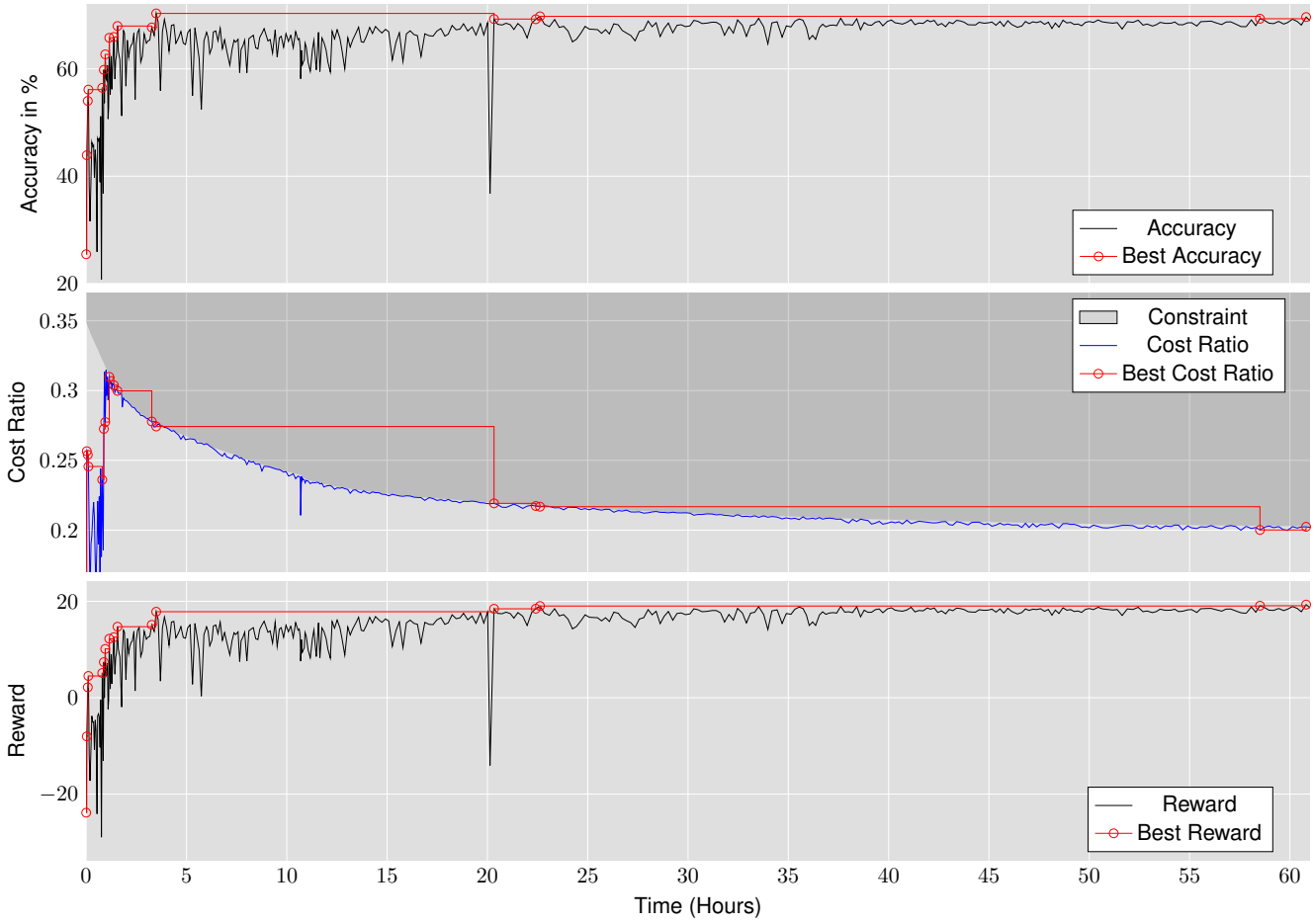

Figure 6. Trajectory of RL agent jointly optimizing ResNet18 for accuracy and latency

breakdown of latencies and tiles for ResNet18 for the baseline implementation as well as the LRMP implementation while optimizing for latency and throughput. Table 8 shows the quantization policies found by LRMP for ResNet18 while optimizing for latency and throughput.

In the baseline case, we observe that the latency of the network is bottlenecked by the first layer, which happens to consume very few tiles. When the layers are replicated for latency optimization (latencyOptim), the total latency is reduced by a factor of 5×, while the latency of the bottleneck layer is reduced by 14× as 13 more copies of that layer are created. In the throughput optimization mode (throughputOptim), the total latency is reduced by a slightly smaller factor of 4.7×, while the latency of the bottleneck layer is reduced by a larger factor of 19× as 18 more copies of that layer are created. This is understandable, because the bottleneck layer is solely responsible for determining throughput, while all layers contribute to latency. It can be observed that LRMP significantly improves tile utlization by balancing the pipeline stages through quantization and

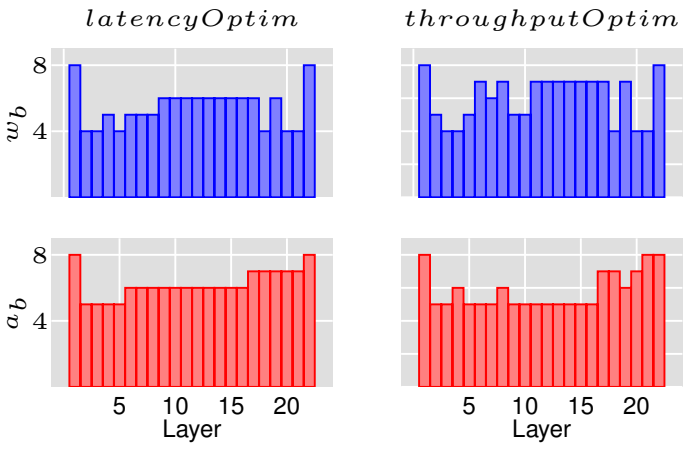

Figure 8. Quantization policies found by LRMP for ResNet18 while optimizing for latency and throughput

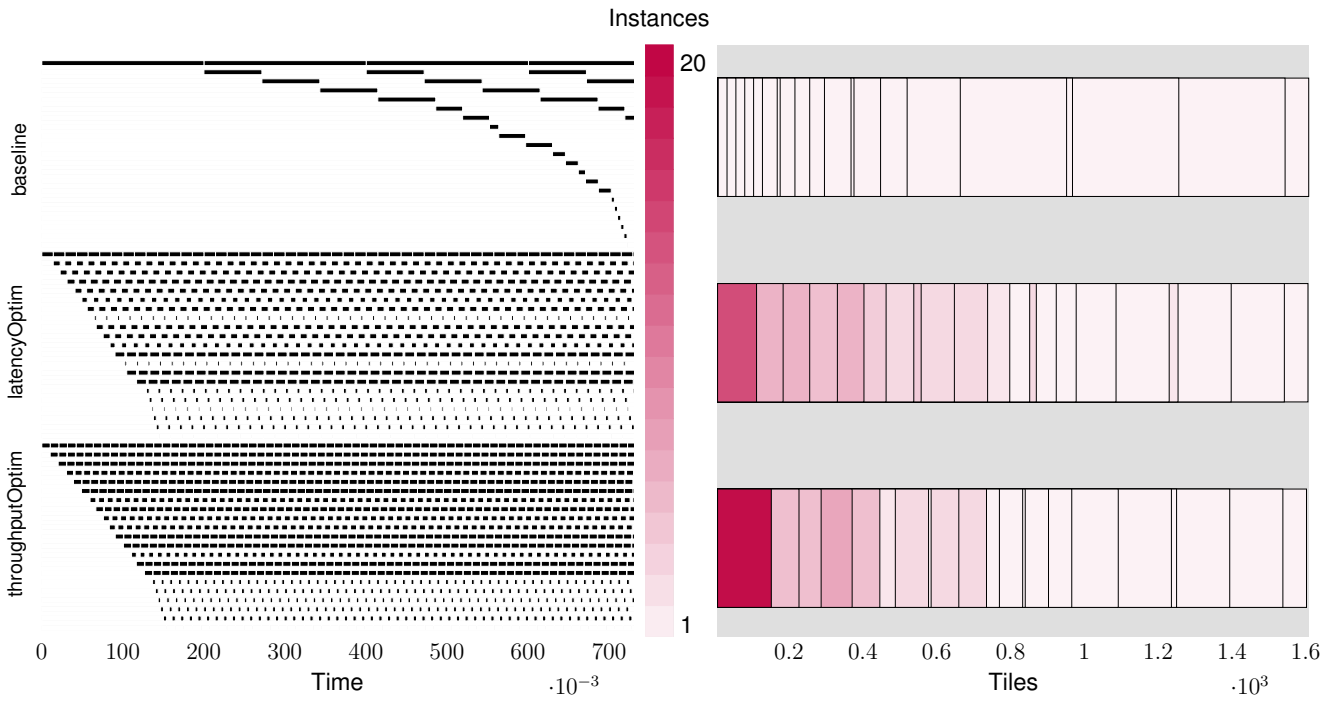

Figure 7. Layer-wise breakdown of latencies and tiles for ResNet18 for the baseline and while optimizing for latency (latencyOptim) and throughput (throughputOptim).

replication, resulting in energy efficiency of 1.54 TOPS/s/W with $throughputOptim$, improving from the baseline 0.2 TOPS/s/W.

6.5 Analysis of sensitivity to chip area

As discussed in Section 5, the layer replication methodology is performed with an area constraint based on the fixed precision baseline i.e., N_{tiles} in the optimization constraints is equal to the number of tiles required by the fixed-precision 8-bit baseline network $baseline\_tiles$. We note that a different design choice, based on the chip area and power budgets, could result in the relaxation or tightening of this tiles constraint.

Fig. 9 shows the sensitivity of the latency improvements achieved by LRMP to different area constraints for the ResNet18 DNN. We perform this analysis by setting N_{tiles} to different ratios of $baseline\_tiles$ and using LRMP to perform only quantization, only replication, and joint quantization and replication. In other words, we study the behavior of LRMP by tightening the tiles constraint below the number of tiles required by the baseline or by relaxing the tiles constraint by making more tiles available in the system, while also using only one of the two optimization dimensions of LRMP.

Because of the model compression naturally achieved by mixed precision, with *only* mixed precision, we achieve 18.5% reduction in latency while using 39% fewer tiles than the baseline. When we employ mixed precision *and* layer replication, we observe latency reductions of 49% while using 35% fewer tiles than the baseline.

We note that layer replication can be performed even without mixed precision, if more tiles are available. We employ *only* layer replication with the baseline ResNet18 and observe 32% reduction in latency while using 5% more tiles than the baseline. It should be noted that when the tiles constraint is tightened, latency reductions are not possible without mixed precision, as there are not enough tiles for even a single copy of all the layers. Also, when all the tiles in the system are used, using mixed precision *and* layer replication gives twice the latency improvement as compared to using only replication.

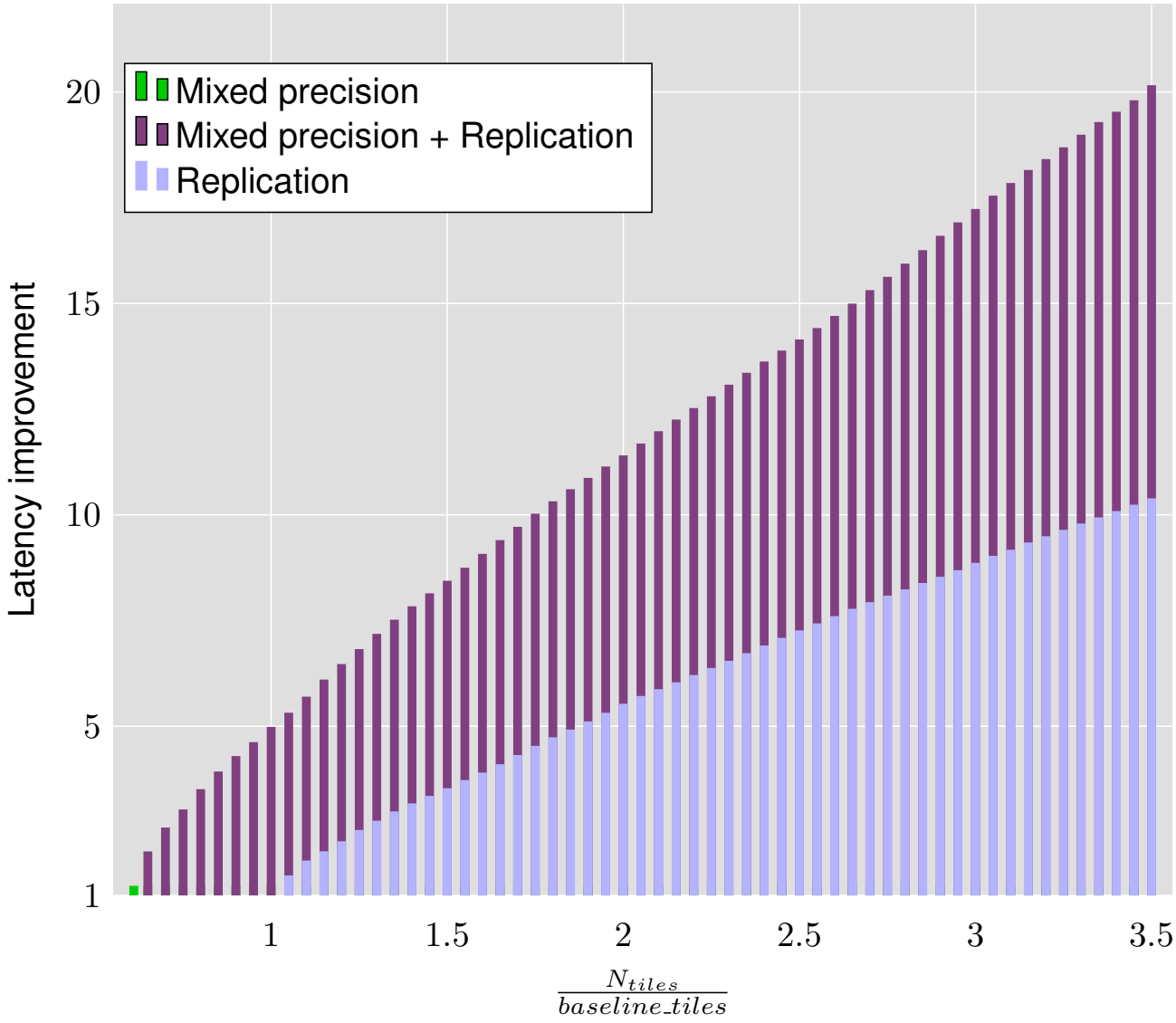

Figure 9. Latency and throughput improvements achieved by the proposed approach on ResNet18 with different area constraints.

7 RELATED WORKS

In this section, we discuss related previous work in the areas of quantization and pruning for DNN implementation on IMC hardware, as well as optimized mapping of DNNs to IMC substrates.

Quantization

Quantization is the process of reducing the number of bits used to represent numbers, which naturally adds distortions in the form of quantization noise. Quantizing weights and activations of neural networks is a common technique to reduce the model size and improve performance of their implementations. Pruning is a technique that removes connections and neurons in a neural network to improve sparsity. These complementary techniques have been widely explored to optimize neural network implementations. Quantization and pruning techniques have also been applied specifically to the context of in-memory computing. Peng et al. (2022) proposed a neural architecture search-based approach to perform mixed precision quantization in a crossbar-aware manner. Kang et al. (2021) proposed a methodology to perform energy-aware quantization using a genetic algorithm. Huang et al. (2021) proposed a methodology that

performs quantization at the tile granularity powered by reinforcement learning. Meng et al. (2021b) proposed a quantization and pruning framework for efficient RRAM IMC implementations.

Mapping Optimization

Mapping neural networks to IMC architectures is a complex problem. Li et al. (2020) proposed an approach to optimize the mapping of multimodal neural networks to IMC hardware to improve throughput. Gopalakrishnan et al. (2020) developed a methodology to design convolutional neural networks that would map better to crossbar architectures. Peng et al. (2019) proposed a weight mapping methodology that would improve data reuse of convolutional layers on crossbars. He et al. (2022) proposed a methodology to replicate layers in a neural network based on area freed-up by optimization of peripheral circuitry.

Optimization of peripheral circuitry

The peripheral circuits of crossbar tiles i.e., the DAC and ADC systems are crucial parts of IMC designs. ADCs contribute to a large portion of the power and area budgets, and are thus, a major bottleneck in the design of IMC systems. Various optimized IMC designs have been proposed that address these bottlenecks. Jiang et al. (2021) discusses an ADC design that implements shifts and adds in the analog domain. Saxena et al. (2022) proposed replacing ADCs with 1-bit sense amplifiers and training neural networks to be tolerant to such aggressive partial sum quantization. He et al. (2022) proposed an approach of decreasing the row parallelism to reduce the area overhead of ADCs and thus, improve the effective density of IMC chips.

To the best of our knowledge, LRMP is the first work that proposes a synergistic methodology that combine the benefits of mixed precision quantization and mapping optimization to jointly optimize the performance and accuracy of IMC-based neural network accelerators. LRMP is also the first work that proposes a linear programming-based approach to perform layer replication in IMC systems. Furthermore, circuit optimizations of tile peripheral are largely complementary to LRMP, and can be used to further improve the performance.

8 CONCLUSION

In-memory computing is a promising technology for accelerating neural networks by performing memory-intensive vector matrix multiplications using analog principles in a memory array. We propose LRMP, a method to synergistically perform layer replication and mixed precision quantization to improve performance of DNNs when mapped to area-constrained IMC accelerators. Our experiments suggest that LRMP can achieve considerable improvements in latency, throughput and energy consumption under iso-utilization and iso-accuracy constraints compared to 8-bit fixed point implementations.

REFERENCES

Ankit, A., Hajj, I. E., Chalamalasetti, S. R., Ndu, G., Foltin, M., Williams, R. S., et al. (2019). Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*. 715–731

Asghari, M., Fathollahi-Fard, A. M., Mirzapour Al-e hashem, S., and Dulebenets, M. A. (2022). Transformation and linearization techniques in optimization: A state-of-the-art survey. *Mathematics* 10, 283

Burr, G. W., Shelby, R. M., Sidler, S., Di Nolfo, C., Jang, J., Boybat, I., et al. (2015). Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Transactions on Electron Devices* 62, 3498–3507

Chang, M., Spetalnick, S. D., Crafton, B., Khwa, W.-S., Chih, Y.-D., Chang, M.-F., et al. (2022). A 40nm 60.64tops/w ecc-capable compute-in-memory/digital 2.25mb/768kb rram/sram system with embedded cortex m3 microprocessor for edge recommendation systems. In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*. vol. 65, 1–3. doi:10.1109/ISSCC42614.2022.9731679

Charan, G., Mohanty, A., Du, X., Krishnan, G., Joshi, R. V., and Cao, Y. (2020). Accurate inference with inaccurate rram devices: A joint algorithm-design solution. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 6, 27–35. doi:10.1109/JXCDC.2020.2987605

Chen, Y., Chen, T., Xu, Z., Sun, N., and Temam, O. (2016). Diannao family: energy-efficient hardware accelerators for machine learning. *Communications of the ACM* 59, 105–112

Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., et al. (2016). Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News* 44, 27–39

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems* 35, 30318–30332

Gao, F., Tziantzioulis, G., and Wentzlaff, D. (2019). Computedram: In-memory compute using off-the-shelf drams. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*. 100–113

Gopalakrishnan, R., Chua, Y., Sun, P., Kumar, A. J. S., and Basu, A. (2020). Hfnet: A cnn architecture co-designed for neuromorphic hardware with a crossbar array of synapses. *Frontiers in Neuroscience* doi:10.3389/fnins.2020.00907

He, K., Chakraborty, I., Wang, C., and Roy, K. (2022). Design space and memory technology co-exploration for in-memory computing based machine learning accelerators. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design* (New York, NY, USA: Association for Computing Machinery), ICCAD '22. doi:10.1145/3508352.3549453

Huang, S., Ankit, A., Silveira, P., Antunes, R., Chalamalasetti, S. R., Hajj, I. E., et al. (2021). Mixed precision quantization for reram-based dnn inference accelerators. *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)* doi:10.1145/3394885.3431554

Jain, S. and Raghunathan, A. (2019). Cxdnn: Hardware-software compensation methods for deep neural networks on resistive crossbar systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 1–23

Jain, S., Ranjan, A., Roy, K., and Raghunathan, A. (2018). Computing in memory with spin-transfer torque magnetic ram. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 470–483. doi:10.1109/TVLSI.2017.2776954

Jain, S., Sengupta, A., Roy, K., and Raghunathan, A. (2020). Rxnn: A framework for evaluating deep neural networks on resistive crossbars. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 326–338

Jain, S., Tsai, H., Chen, C.-T., Muralidhar, R., Boybat, I., Frank, M. M., et al. (2023). A heterogeneous and programmable compute-in-memory accelerator architecture for analog-ai using dense 2-d mesh. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31, 114–127. doi:10.1109/TVLSI.2022.3221390

Jiang, H., Li, W., Huang, S., Cosemans, S., Cosemans, S., Cosemans, S., et al. (2021). Analog-to-digital converter design exploration for compute-in-memory accelerators. *IEEE Design and Test of Computers* 38, 1–8

Jouppi, N., Young, C., Patil, N., and Patterson, D. (2018). Motivation for and evaluation of the first tensor processing unit. *ieee Micro* 38, 10–19

Kang, B., Lu, A., Long, Y., Kim, D. H., Yu, S., Yu, S., et al. (2021). Genetic algorithm based energy-aware cnn quantization for processing-in-memory architecture. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* doi:10.1109/jetcas.2021.3127129

Kang, M., Gonugondla, S. K., and Shanbhag, N. R. (2020). Deep in-memory architectures in sram: An analog approach to approximate computing. *Proceedings of the IEEE* 108, 2251–2275

Khaddam-Aljameh, R., Stanisavljevic, M., Mas, J. F., Karunaratne, G., Braendli, M., Liu, F., et al. (2021). Hermes core–a 14nm cmos and pcm-based in-memory compute core using an array of 300ps/lsb linearized cco-based adcs and local digital processing. In *2021 Symposium on VLSI Circuits* (IEEE), 1–2

Li, B., Wang, Y., and Chen, Y. (2020). Hitm. *Proceedings of the 39th International Conference on Computer-Aided Design* doi:10.1145/3400302.3415663

Li, W., Han, Y., and Chen, X. (2023a). Mathematical framework for optimizing crossbar allocation for reram-based cnn accelerators. *ACM Trans. Des. Autom. Electron. Syst.* 29. doi:10.1145/3631523

Li, X., Liu, Y., Lian, L., Yang, H., Dong, Z., Kang, D., et al. (2023b). Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 17535–17545

Liang, T., Glossner, J., Wang, L., Shi, S., and Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461, 370–403

Lie, S. (2022). Cerebras architecture deep dive: First look inside the hw/sw co-design for deep learning: Cerebras systems. In *2022 IEEE Hot Chips 34 Symposium (HCS)* (IEEE Computer Society), 1–34

Lu, A., Peng, X., Li, W., Jiang, H., and Yu, S. (2021). Neurosim simulator for compute-in-memory hardware accelerator: Validation and benchmark. *Frontiers in artificial intelligence* 4, 659060

Meng, J., Shim, W., Yang, L., Yeo, I., Fan, D., Yu, S., et al. (2021a). Temperature-resilient rram-based in-memory computing for dnn inference. *IEEE Micro* 42, 89–98

Meng, J., Yang, L., Peng, X., Yu, S., Fan, D., and sun Seo, J. (2021b). Structured pruning of rram crossbars for efficient in-memory computing acceleration of deep neural networks. *IEEE Transactions on Circuits and Systems Ii-express Briefs* doi:10.1109/tcsii.2021.3069011

Narayanan, P., Ambrogio, S., Okazaki, A., Hosokawa, K., Tsai, H., Nomura, A., et al. (2021). Fully on-chip mac at 14 nm enabled by accurate row-wise programming of pcm-based weights and parallel vector-transport in duration-format. *IEEE Transactions on Electron Devices* 68, 6629–6636

Peng, J., Liu, H., Zhao, Z., Li, Z., Liu, S., and Li, Q. (2022). Cmq: Crossbar-aware neural network mixed-precision quantization via differentiable architecture search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* doi:10.1109/tcad.2022.3197495

Peng, X., Liu, R., and Yu, S. (2019). Optimizing weight mapping and data flow for convolutional neural networks on rram based processing-in-memory architecture. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. doi:10.1109/ISCAS.2019.8702715

Perez, E., Mahadevaiah, M. K., Quesada, E. P.-B., and Wenger, C. (2021). Variability and energy consumption tradeoffs in multilevel programming of rram arrays. *IEEE Transactions on Electron Devices* 68, 2693–2698. doi:10.1109/TED.2021.3072868

Rasch, M. J., Gokmen, T., Rigotti, M., and Haensch, W. (2019). Rapa-convnets: Modified convolutional networks for accelerated training on architectures with analog arrays. *Frontiers in Neuroscience* 13, 753

Roy, S., Sridharan, S., Jain, S., and Raghunathan, A. (2021). Txsim: Modeling training of deep neural networks on resistive crossbar systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 730–738

Saxena, U., Chakraborty, I., and Roy, K. (2022). Towards adc-less compute-in-memory accelerators for energy efficient deep learning. *Design, Automation and Test in Europe* doi:10.23919/date54114.2022.9774573

Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J. P., Hu, M., et al. (2016). Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 14–26

Shim, W., Meng, J., Peng, X., Seo, J.-s., and Yu, S. (2021). Impact of multilevel retention characteristics on rram based dnn inference engine. In *2021 IEEE International Reliability Physics Symposium (IRPS)*. 1–4. doi:10.1109/IRPS46558.2021.9405210

Song, L., Qian, X., Li, H., and Chen, Y. (2017). Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 541–552. doi:10.1109/HPCA.2017.55

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. (2019). Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8612–8620

Yan, H., Cherian, H. R., Ahn, E. C., and Duan, L. (2018). Celia: A device and architecture co-design framework for stt-mram-based deep learning acceleration. In *Proceedings of the 2018 International Conference on Supercomputing*. 149–159

Yin, S., Jiang, Z., Seo, J.-S., and Seok, M. (2020). Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits* 55, 1733–1743

Zhang, J., Wang, Z., and Verma, N. (2017). In-memory computation of a machine-learning classifier in a standard 6t sram array. *IEEE Journal of Solid-State Circuits* 52, 915–924

Zhu, Z., Sun, H., Lin, Y., Dai, G., Xia, L., Han, S., et al. (2019). A configurable multi-precision cnn computing framework based on single bit rram. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6