2025 Design, Automation & Test in Europe Conference (DATE 2025)

Odin: Learning to Optimize Operation Unit Configuration for Energy-efficient DNN Inferencing

Gaurav Narang, Janardhan Rao Doppa, Partha Pratim Pande
School of EECS, Washington State University, Pullman, WA, USA
{gaurav.narang, jana.doppa, pande}@wsu.edu

***Abstract*—ReRAM-based Processing-In-Memory (PIM) architectures enable energy-efficient Deep Neural Network (DNN) inferencing. However, ReRAM crossbars suffer from various non-idealities that affect the overall inferencing accuracy. To address that, the matrix-vector-multiplication (MVM) operations are computed by activating a subset of the full crossbar, referred to as Operation Unit (OU). However, OU configurations vary with the neural layers' features such as sparsity, kernel size, and their impact on predictive accuracy. In this paper, we consider the problem of learning appropriate layer-wise OU configurations in ReRAM crossbars for unseen DNNs at runtime such that the performance is maximized without loss in predictive accuracy. We develop a machine learning (ML) based framework called *Odin*, which selects the OU sizes for different neural layers as a function of the neural layer features and time-dependent ReRAM conductance drift. Our experimental results demonstrate that the energy-delay-product (EDP) is reduced by up to 8.7× over state-of-the-art homogeneous OU configurations without compromising predictive accuracy.**

Keywords— ReRAM, Deep Neural Networks, Operation Units, Online learning, Processing-in-Memory, Conductance drift

I. Introduction

Deep Neural Networks (DNNs) have become the primary engines of artificial intelligence, driving remarkable advancements across numerous real-world applications, including natural language processing, image recognition, and healthcare [1] [2] [3] [4]. However, substantial computational and memory demands of DNNs pose significant challenges to the performance and energy efficiency of traditional CPU/GPU-based platforms. Processing-in-Memory (PIM) has emerged as a promising solution, offering high-performance and energy-efficient acceleration of DNN inference/training tasks [5] [6] [7]. Among different non-volatile memory (NVM)-based devices, ReRAMs have emerged as one of the popular choices to perform matrix-vector multiplication (MVM) (i.e. the key computational kernel of DNNs) efficiently [8] [9].

ReRAM crossbars are vulnerable to errors arising due to non-idealities such as thermal noise, IR-drop, conductance drift, etc. [10] [11] [12]. These non-idealities change the programmed conductance values corresponding to DNN weights, thus leading to a loss in predictive accuracy [13]. To tackle such non-idealities, MVM operations are generally executed at a much smaller level of granularity than a full crossbar, referred to as an *Operation Unit (OU)* [14] [15]. For example, only 16 wordlines (WLs) and 16 bitlines (BLs) are activated concurrently in a 128×128 crossbar for reliable operation [16]. OUs also allow us to exploit sparsity in DNN layers by skipping the rows with zeros in an OU to improve performance and energy-efficiency. However, choosing the appropriate OU size is a non-trivial task. A relatively bigger OU size gives rise to more IR-drop along the WL and BL, leading to a loss in predictive accuracy. On the other hand, a smaller OU size can lead to higher latency and energy consumption, as it takes multiple cycles to compute the entire crossbar with the fine-grained OUs. Further, DNN layers can vary greatly in terms of the number of weights, kernel sizes, levels of sparsity, and their impact on predictive accuracy. However, current approaches do not take these layer-specific characteristics into account when determining the OU size. Instead, the OU size is kept constant across all DNN layers, which can limit the achievable performance gain and energy efficiency. Hence, it is potentially beneficial to optimize OU configurations based on the layer characteristics of a DNN.

ReRAM crossbars have limited data retention capabilities since the on-state conductance reduces over time [17]. Due to conductance drift, the impact of non-idealities on the accuracy varies over time. An OU size optimized for negligible accuracy loss initially may lead to degradation in accuracy over time. Device reprogramming is a common approach to counteract conductance drift, but it can introduce high energy overhead [17] [18]. Therefore, minimizing reprogramming energy is crucial. Reducing the OU size over time offers a way to address the combined impact of device and crossbar non-idealities, potentially lowering the reprogramming energy cost. *Existing work so far has relied on static OU-based computation* and has not explored the benefits of dynamically varying OU configurations to mitigate the effect of device non-idealities. Further, the same OU configuration may not be optimal for every DNN model. An OU configuration computed *offline* for a known DNN model at design time may not be optimal for unseen DNNs at runtime. Hence, there is a strong need for an *online learning* approach, which can adapt to new DNNs and time-dependent conductance drift by learning optimized layer-wise OU configurations at runtime.

In this work, we propose a novel online learning approach called *Odin* to determine optimized OU configurations for inferencing with unseen DNNs. Odin employs neural layer features of the DNN model such as sparsity, kernel size, and inference time elapsed from initial DNN weight programming (input) to predict the layer-wise optimized OU sizes (output) for each DNN inference run. Odin performs a resource-bounded search over the candidate OU configurations using the energy, latency, and non-ideality estimations to identify the OU configuration with minimum energy-delay-product (EDP) and negligible inference accuracy loss. This serves as a supervised training example for online learning. The key contributions of this work are:

- We propose an online learning framework *Odin* that learns to optimize OU configurations for inferencing with unseen DNNs at runtime.
- We propose a novel layer-wise OU-based computation framework where the OU size varies with the characteristics of the DNN neural layers, such as sparsity and kernel size, as well as the time-dependent conductance drift of ReRAM devices.
- Our experimental results demonstrate that Odin reduces the energy-delay-product (EDP) by up to 8.7× compared

This work was supported by the US National Science Foundation grant CSR-2308530 and by the Army Research Office grant ARO-W911NF-24-1-0240.

978-3-9826741-0-0/DATE25/© 2025 EDAA

to the state-of-the-art homogeneous OU-based configurations on popular DNNs and vision transformer models for classification tasks using CIFAR-10, CIFAR-100, and TinyImageNet datasets.

II. Related Prior Work

Several ReRAM-based PIM accelerators have been proposed in the literature [8] [9]. However, ReRAM crossbars are vulnerable to errors arising due to device and crossbar non-idealities such as thermal noise, IR-drop, programming errors, conductance drift, etc. [10] [11] [19] [20]. These non-idealities change the conductance values corresponding to actual DNN weights. This leads to a loss in predictive accuracy. Prior work has shown that the non-idealities increase as more WLs and BLs are activated concurrently in a crossbar [19] [21]. To maintain predictive accuracy, a small number of WLs and BLs of a crossbar are activated in a compute cycle. The fine-grained OU configuration that is being activated in one cycle is referred to as an Operation Unit (OU) [14] [22]. Prior work has demonstrated that OU-based computation can exploit weight and activation sparsity effectively, leading to high performance and significant energy savings [14] [23]. Weight compression has been done at the OU level in either row or column dimensions of a crossbar, requiring the storage of input and output indices to fetch the correct inputs for compressed weights [14] [16]. However, input and output indices are computed offline and stored in a buffer prior to DNN workload execution. Further, none of the existing works have explored the layer characteristics of the DNNs while determining the OU size, i.e., OU size remains constant across all the DNN layers. Due to time-dependent conductance drift, optimized OU configurations may vary over time, requiring unlimited storage for input and output indices. Prior works have enhanced the compression further by reordering the filters at the OU level, and by using weight, input pattern repetitions, etc. [24] [25]. However, the computation of optimized filter reordering, weight and input repetition patterns is done offline for a given OU configuration and stored in the buffer, prior to DNN inferencing. Since filter reordering, weight and input repetition patterns will differ for each DNN, these techniques will lead to performance overhead to adapt to unseen DNNs at runtime. In contrast to prior work, we propose a novel low-overhead online learning framework that learns to optimize layer-wise OU sizes for unseen DNN models depending on the neural layer, device, and crossbar characteristics.

III. Odin Online Learning Framework

Problem Setup. Without loss of generality, we consider the j^{th} neural layer of a DNN is computed with an OU of size (height R_j × width C_j) on a crossbar of size $c \times c$, where R_j, C_j are integer values $\in [1, c]$. As an example, R_j, C_j can be any integer values between 1 and 128 for a crossbar size of 128×128. We are given a set of DNNs and input datasets (e.g., images) to create an offline OU policy to effectively bootstrap the online learning process. It should be noted that estimating predictive inference accuracy of an unseen DNN model without prior knowledge of the true labels is not feasible at runtime. Hence, we use non-ideality in ReRAMs and crossbar as a surrogate for predictive accuracy and constrain it by an input threshold η (say, 0.5%). Our aim is to learn to optimize layer-wise OU configurations to minimize EDP without loss in predictive accuracy while inferencing with any new DNN models. In this work, we consider the OU configuration policy

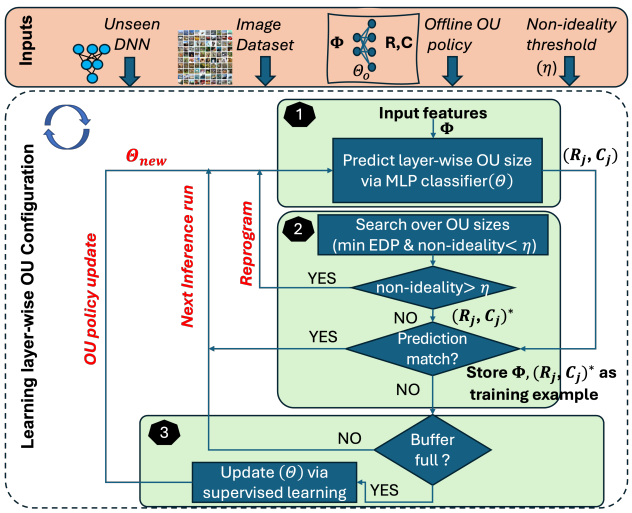


Fig. 1: High level overview of Odin framework and key elements.

(π) represented as a function of the neural layer, ReRAM device, and crossbar characteristics with parameters $\theta \in \mathbb{R}$ (e.g., multi-layer perceptron (MLP)). The online OU configuration policy takes into account the neural layer features (e.g., sparsity, kernel size) and inference time elapsed from initial DNN weight programming and produces a multi-output decision $(R_j \times C_j)$ for the j^{th} neural layer. Our goal is to learn the OU configuration policy parameters θ to adapt to unseen DNNs at runtime such that EDP is minimized without loss in predictive accuracy.

Overview of Odin. We initialize the OU configuration policy (created offline) using known DNNs at the design time. Since the OU configuration policy is a mapping from neural layer features to OU size for each DNN layer, this knowledge can be useful with varying degree even for unseen DNNs at runtime. In each inference run, Odin performs the following sequence of algorithmic steps. First, Odin uses the most up to date OU policy to predict OU configuration decision $(R_j \times C_j)$ for the j^{th} incoming neural layer. Next, to adapt to the unseen DNN, we perform a resource-bound search (guided by OU-based energy, latency, and non-ideality analytical models) to determine the optimized (best) layer-wise OU size $(R_j \times C_j)^*$. The best configuration $(R_j \times C_j)^*$ is the one with minimum EDP and negligible loss in accuracy (measured by the change in stored conductance value due to non-idealities). If there is no OU size that ensures negligible loss in accuracy (i.e., non-ideality > η), ReRAMs are reprogrammed before the next inference run. In case the reprogramming is not required, we compare the policy decision $(R_j \times C_j)$ and the best decision $(R_j \times C_j)^*$. If they do not match, the best uncovered decision is used for future training to update the policy to adapt to unseen DNN. The newly created training examples of neural layer features and inference time (input (Φ)) and best OU configuration $(R_j \times C_j)^*$ (output) are added to a buffer. If the buffer is full, Odin updates the parameters θ of the OU configuration policy π using the aggregated training data. Fig. 1 shows the high-level overview, and Algorithm 1 shows the pseudocode of the Odin framework. Next, we describe the details of the key elements of the framework.

A. Features and Policy Representation

We consider the OU configuration policy represented as functions of the input features, i.e., neural layer, ReRAM device, and crossbar characteristics. The four input features Φ include the neural layer characteristics (neural layer

identifier (Φ_1), sparsity (Φ_2), kernel size (Φ_3)) and inference time (Φ_4) elapsed from the time when the device was first programmed. The *neural layer identifier* captures the neural layers' varying impact on predictive accuracy. As an example, the initial neural layers extract key characteristics of an input image for classification tasks. Hence, non-idealities of crossbars executing the initial neural layers have a higher impact on predictive accuracy than that in latter neural layers [19] [26]. *Sparsity* and *kernel size* represent the physical location of zero weights and the computational requirements of each neural layer respectively. These features determine the number of OU compute cycles for a neural layer and thus DNN's performance in terms of latency and energy. Lastly, the duration of the total inferencing process (*inference time*) determines the impact of conductance drift on accuracy.

DNNs can vary in terms of number of channels, filters, input and output feature maps, etc., depending on the type of user application. Hence, it is not scalable to store optimized OU configurations for unlimited configurations of DNN models. Further, the optimized OU configurations may change for a particular DNN over time due to time-varying conductance drift. Thus, we employ a neural network (NN)-based policy $\pi(\Phi, \Theta)$ with parameters Θ as a parametric function of input features Φ (DNN characteristics and inference time) [27]. We use features Φ and corresponding optimized OU configurations of known DNNs as training data to train an offline policy. Specifically, we use a multi-output MLP classifier to learn to predict the OU configuration ($R_j \times C_j$) for the j^{th} neural layer of the unseen DNN [28]. A key advantage of such an NN-based policy is that it can approximate complex functions and adapt to unseen DNNs at runtime with low computational and storage overheads.

B. Supervised Training Data

At runtime, we predict layer-wise OU configuration ($R_j \times C_j$) for j^{th} neural layer using the current (offline or updated) OU policy $\pi(\Phi, \Theta)$ (Algorithm 1, line 5). Our goal is to adapt the OU policy to incoming DNNs at runtime. This requires supervised training data to update the OU policy. We use analytical OU-based energy, latency, and nonideality models to determine the best OU configurations to create new training data. Next, we explain these analytical models.

Latency: ADC is the critical part of the pipeline in ReRAM-based PIM accelerators [29]. ADC's sensing delay is proportional to its bit resolution [30]. ADC's bit precision is determined by the number of activated WLs (R_j) and is proportional to $\log_2 R_j$ [8]. Overall latency to compute a neural layer is given by (1), where C_j is the OU width and OU_j is the number of OU cycles required to execute the j^{th} neural layer. Note that the number of OU cycles in a crossbar depends on the neural layer's sparsity.

$$\text{Latency} \cong C_j \cdot \log_2 R_j \cdot OU_j \quad (1)$$

Energy: ADC energy consumption increases with the number of WLs (R_j) in each OU and the number of OU cycles to execute a neural layer. The number of OU cycles (OU_j) is a function of sparsity, OU size ($R_j \times C_j$), and the number of crossbars ($Xbar_j$) needed to map the neural layer. Thus, the inferencing energy of a neural layer is given by (2).

$$\text{Energy} \cong Xbar_j \cdot \log_2 R_j \cdot R_j \cdot C_j \cdot OU_j \quad (2)$$

Non-Ideality: ReRAM crossbars are vulnerable to non-idealities such as IR-drop and conductance drift. IR-drop increases as more WLs and BLs are activated concurrently.

Algorithm 1. Learning to optimize OU configurations (Odin)

Input: input (image) dataset,
APP = inferencing task (unseen DNN) at runtime,
η= non-ideality factor (NF) threshold,
$\pi(\Phi, \Theta_0)$= neural network-based OU configuration policy with parameters Θ_0 trained offline using known DNNs at design time

Output: parameters of the OU policy optimized for *APP*

1: **Initialize**: b_0 = empty buffer; Θ_0=parameters of the OU configuration policy created offline and $t = t_o$
2: **Repeat** for each inference run**:**
3: **For** each neural layer j:
4: $\Phi \leftarrow$Estimate features /* Layer_id, Sparsity, Kernel size, Inference time */
5: $(R, C) \leftarrow$Predict OU size via OU policy $\pi(\Phi, \Theta)$
6: $(R, C)^* \leftarrow$Search for OU size that minimizes EDP such that $Nonideality < \eta$ /* Eq. (1)-(4) */
7: **If** $Nonideality > \eta$ for all OU sizes: /* No OU size satisfies nonideality constraint*/
8: $Reprogram\ DNN\ weights, t \leftarrow t_o$
9: **Else if** $(R, C) \neq (R, C)^*$:
10: Store training example in a buffer. $b_{t+1} = b_t\ U\ \{\Phi, (R, C)^*\}$
11: **If** buffer is full; reset the buffer. Supervised learning on training examples to update the parameters of OU policy to Θ_{t+1}
12: $t = t + 1$
13: **Until end of *APP* inferencing**
14: **return** parameters of the OU policy optimized for *APP*

IR-drop also varies with the ratio of crossbar wire resistance (R_{wire}) and on-state conductance (G_{ON}) [13]. The on-state conductance (G_{ON}) reduces due to conductance drift, given by (3), where v is drift co-efficient, t_o is the initial device programming time, and t is the time elapsed while inferencing [17]. As a result, the severity of IR-drop increases with inferencing time.

$$G_{drift}(t) = G_{ON} \cdot \left(\frac{t}{t_o}\right)^{-v} \quad (3)$$

These non-idealities lead to change in the stored conductance of the ReRAM cells. The change in conductance (ΔG) executing neural layer j at inference time t is given by (4). Thus, non-ideality depends on the layer-wise OU sizes, inference time elapsed after device programming, device properties (e.g., G_{ON}, v), and crossbar properties (e.g., R_{wire}).

$$\Delta G = \left| G_{ON} - \left\{ \frac{1}{\frac{1}{G_{drift}(t)} + R_{wire} \cdot (R_j + C_j)} \right\} \right| \quad (4)$$

Resource-bounded search. Using the above-defined analytical models, we search for the best OU configuration $(R_j \times C_j)^*$ (Algorithm 1, line 6). We start the local search in the neighborhood of policy's decision ($R_j \times C_j$) by setting a maximum allowable search count K (e.g., 3). We evaluate the EDP and ΔG of the OU configurations that are within K steps from ($R_j \times C_j$) by changing R_j, C_j by 1 (i.e., ± 1). We update the best configuration $(R_j \times C_j)^*$ whenever a configuration in the neighborhood search outperforms the best configuration so far to minimize EDP such that $\Delta G < \eta$.

From (4), we observe that non-ideality ΔG can be reduced if the OU size ($R_j \times C_j$) is reduced as the conductance drift increases. However, if there is no OU size that ensures negligible loss in accuracy i.e., $\Delta G > \eta$, ReRAMs are reprogrammed before the next inference run (Algorithm 1, lines 7 and 8). Otherwise, if a best OU configuration exists (minimum EDP and $\Delta G < \eta$), it is compared with the one predicted by the current policy. If the OU configuration

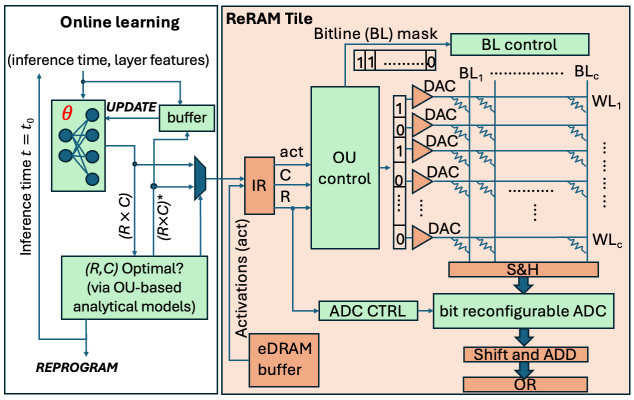


Fig. 2: Dataflow of layer-wise OU-based computation for unseen DNNs.

chosen by the current policy is different from $(R_j \times C_j)^*$, then we store $(R_j \times C_j)^*$ along with input features Φ in a buffer (Algorithm 1, lines 9 and 10). This data serves as the supervised training data which is used for online learning to update the parameters Θ of OU policy.

C. *Online Policy Update*

Next, we update the OU configuration policy parameters as a function of new training examples. We aggregate the training examples until the buffer is full (Algorithm 1, line 11). The size of the buffer is important since it determines the training accuracy and storage overhead. Once the buffer is full, we update the OU policy using supervised learning on the stored training examples. We use this updated OU policy for the subsequent inference runs.

IV. Odin-enabled Architecture

Fig. 2 shows the architecture of the proposed layer-wise OU-based computation and online learning to optimize OU sizes for unseen DNNs at runtime. In each inference run, an MLP classifier-based policy predicts layer-wise OU sizes $(R_j \times C_j)$. OU-based analytical models (using (1) to (4)) are used to determine the best OU size $(R_j \times C_j)^*$ for the unseen DNN such that EDP is minimized with $\Delta G < \eta$. If $\Delta G > \eta$ for all OU sizes, then *REPROGRAM* pulse is sent to ReRAM reprogramming logic, and inference time is reset. Otherwise, if the best OU size matches the policy decision, we continue inferencing with the policy decision. The OU size is stored in the input register (IR). If the policy decision does not match with the best OU size, then input features and the best OU size are stored in an on-chip buffer (as training data to update the policy). If the buffer is full, *UPDATE* pulse is sent to update the policy parameters. We store 50 training examples for supervised learning to ensure training convergence of OU policy (requiring 0.35 KB storage). The OU size prediction introduces negligible overhead, quantified in Section V.

The input activations are fetched from eDRAM buffer and stored in the IR. The OU controller forms virtual OU units by activating the appropriate WL and BL in a crossbar, corresponding to the layer-wise OU size and input activations. Cell currents are read by the S&H units and fed to the ADCs. The architecture utilizes reconfigurable ADCs to support layer-wise OU configurations [29]. The number of bits (precision) of the ADC can be lowered by disabling the lower LSB bits of the ADC (controlled by OU height R_j). The output from the ADC is stored in the output register (OR) and passed to the subsequent neural layer.

TABLE 1. PIM Architecture Specifications

Tile Configuration (1.2 GHz, 32nm, 0.28 mm^2)		
Component	**Specification**	**Area (mm^2)**
eDRAM buffer	size:64KB	0.083
eDRAM bus	buswidth:384	0.09
Router	flit:32, port 8	0.0375
Sigmoid, S+A, Maxpool	number:2,96,1	0.0038
OR, IR	size:3KB, 2KB	0.0282
OU Control	number:1	0.0048
ADC (with control)	number:96; reconfigurable precision 3 to 6 bits	0.03
DAC, S+H	number:96×128	0.0025
Memristor array	number:96, size:128×128, bits/cell:2, OU size: varying	0.0024

TABLE II. Parameters of ReRAM Crossbar System

Parameter	Description	Value
R_{wire}	Crossbar wire resistance	1 ohm
G_{ON}/G_{OFF}	ON/OFFF state conductance	333/0.33 μS
v	Drift coefficient	0.2 s^{-1}

V. Experimental Setup and Analysis of Results

A. *Experimental Setup*

PIM Architecture: Without loss of generality, we consider a PIM architecture consisting of 36 ReRAM-based processing elements (PEs) connected through a conventional mesh-based NoC. The overall methodology is valid for any other system size and NoC configuration. Each PE has 4 tiles, and each tile has 96 ReRAM crossbars of size 128×128 [8]. Table I shows the ReRAM tile configuration. A dedicated ReRAM digital PIM core is used for high precision (32-bit floating point) weight gradient computation to update the OU policy parameters, following prior work [31]. The hardware associated with the online learning controller and the OU controller are synthesized in the 32 nm technology node. The energy and latency of the ReRAM-based PEs, including peripheral circuits such as ADCs, buffers, and nonlinear activation units (ReLU), are obtained via NeuroSim [32]. The inference accuracy is obtained using PytorX [33]. The impact of non-idealities of ReRAMs and crossbar on the accuracy are incorporated in PytorX using the parameters listed in Table II [33] [13]. The non-ideality threshold η is set to 0.5% to ensure negligible accuracy loss, following prior work [18].

OU policy representation: For a ReRAM crossbar of size 128×128, we consider the OU size (R_j, C_j) to be constrained by 2^L, where L is an integer value between 2 and 7 (i.e., 6 discrete values). The MLP consists of one input layer (4 neurons) with the ReLU activation and two separate output layers (6 neurons each) with the softmax activation.

DNN models and Datasets: We evaluate the performance of the proposed Odin-enabled PIM architecture considering various DNNs, vision transformer (ViT) model, and datasets. The DNNs used in our evaluation include ResNet18, VGG11, GoogLeNet, DenseNet121, ViT on CIFAR-10 dataset, ResNet34, VGG16 on CIFAR-100 dataset, and ResNet50, VGG19 on TinyImageNet dataset. To demonstrate the effectiveness of online learning on one type of DNN, the offline policy is constructed using $(N-1)$ DNNs. As an example, to evaluate VGG (linear) DNNs, offline OU policy is learnt from ResNets (residual), DenseNets (dense), ViT (encoder-decoder), etc., and Odin learns to optimize OU

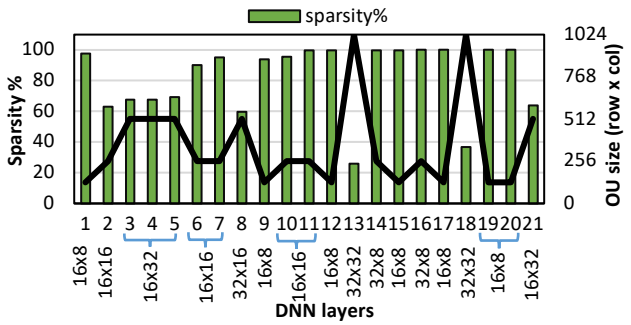


Fig. 3: Layer-wise OU size and weight sparsity for ResNet18 neural layers (including skip connections) on CIFAR-10 dataset.

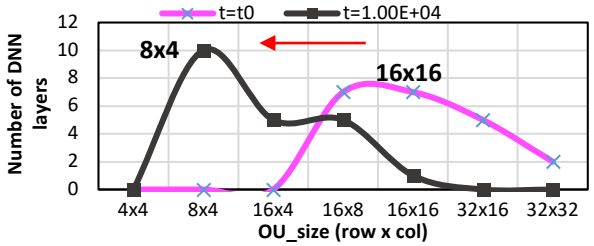
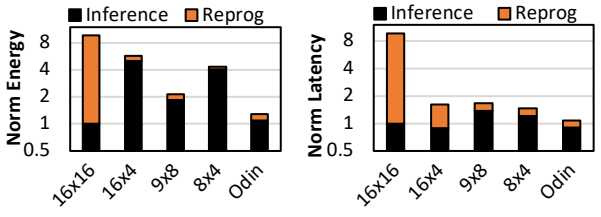


Fig. 4: OU size distribution shift under the impact of conductance drift at t seconds for ResNet18 on CIFAR-10 dataset.

configurations for VGG models at runtime. The offline policy is constructed using up to 500 training examples (comprising of neural layer features and optimized OU configurations of known DNNs). We implement a crossbar-aware weight and activation pruning to obtain highly sparse pre-trained DNN models [29]. However, the proposed framework is applicable for any other pruning method.

B. *Online Learning Performance*

Fig. 3 shows layer-wise OU size (represented as $R_j \times C_j$ product) along with weight sparsity (%) for ResNet18 at time instance t_o (start of the inferencing process). Since the initial neural layers capture key features of the input images, they have a relatively higher impact on the accuracy compared to the later layers. Thus, Odin chooses relatively smaller OUs (e.g., 16×8) for the initial layers to reduce the IR-drop, as shown in Fig. 3. This helps in maintaining the DNN predictive accuracy. On the contrary, Odin chooses coarse-grained OUs (e.g., 32×32) for the later neural layers with lower sparsity (layers 13 and 18). This improves the performance by reducing the number of OU cycles required to execute a neural layer with low sparsity. In addition to neural layer features, OU sizes vary with time as well. Fig. 4 shows the OU size distribution shift for all the neural layers under the impact of non-idealities for ResNet18 as a representative example. Here, by '*Number of DNN layers*' we imply how many DNN layers have the specific OU configurations. As shown in Fig. 4, the peak of the OU-size distribution shifts to the left towards relatively fine-grained



(a) (b)

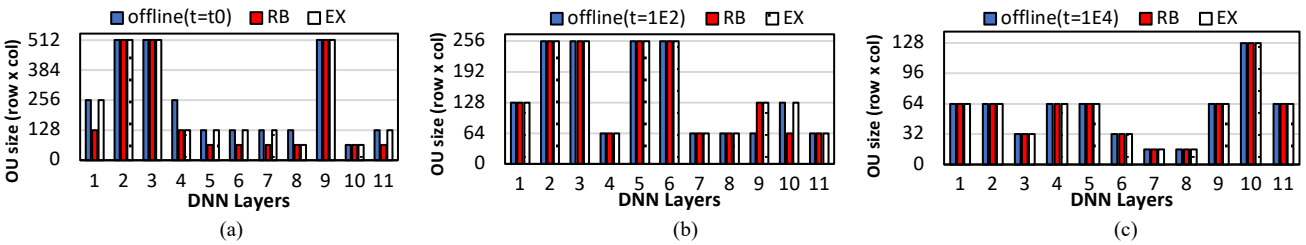
Fig. 6: Comparison of various OU configurations in terms of (a) energy, and (b) latency for VGG11 (CIFAR-10) normalized with respect to (16 × 16) OUs inferencing energy and latency.

OUs such as (8×4) over time. This helps in reducing the crossbar non-idealities (IR-drop) as the conductance drift increases, thereby minimizing the loss in predictive accuracy. Thus, there is a need for learning to optimize OU configurations as a function of neural layer characteristics and inference time.

Next, we demonstrate Odin's ability to determine the OU configurations for unseen DNNs at runtime. Figs. 5(a)-5(c) show the layer-wise OU configurations (represented as $R_j \times C_j$ product) chosen by the online learning and optimized offline for VGG11 as a representative example. Fig. 5 also compares the best OU configuration $(R_j, C_j)^*$ determined via exhaustive (EX) and resource-bound (RB) searches. As shown in Fig. 5(b), as the OU policy adapts to unseen DNN by $t = 1.00\text{E+}02$s, the OU configuration chosen by RB closely follows the true OU configuration determined offline. As shown in Fig. 5(a), the OU configurations chosen via EX are comparatively closer to the offline counterpart than RB initially (e.g., DNN layers 5, 6, and 7). Thus, EX can generate higher quality online policy compared to RB. However, the comparator logic associated with the search for $(R_j, C_j)^*$ introduces significant timing overhead, which increases logarithmically with the OU configuration search space. As an example, EX leads to a ~3 × higher timing overhead compared to RB (search step parameter $K = 3$, discussed in Section III) for 6 discrete levels of R_j, C_j each (36 OU configurations). This highlights the merit of RB search for low-overhead online learning and its potential for scalability.

C. *Comparative Performance Evaluation*

Reduction in reprogramming: Fig. 6 compares Odin and various homogeneous OU configurations determined offline in terms of energy and latency. Following homogeneous OU configurations from prior work are used for comparison with Odin: (16×16), (16×4), (9×8), and (8×4) [16] [24] [34]. Computation with coarse-grained OUs results in higher crossbar non-idealities compared to fine-grained OUs. As a result, coarse-grained OUs require relatively more frequent reprogramming as the conductance drift increases to maintain predictive accuracy. As an example, (16×16) requires device reprogramming 43 times from time t_0 to



(a) (b) (c)

Fig. 5: Comparison of layer-wise OU configurations for known DNN (offline) vs. learnt online by Odin (via RB=resource-bound and EX=exhaustive search) at time (a) $t = t_o$, (b) t =1.00E+02s, and (c) t =1.00E+04s for an unseen DNN VGG11 on CIFAR10 dataset.

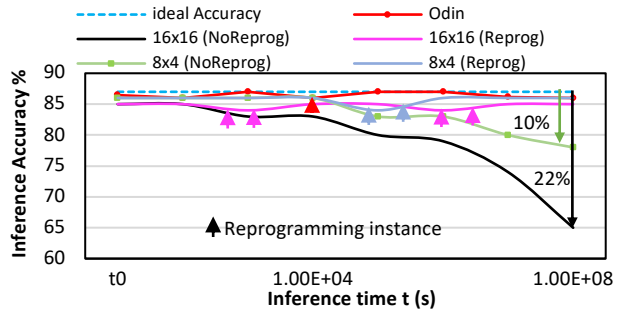


Fig. 7: Inference accuracy (%) over the inference runs for VGG11 (CIFAR-10) utilizing various OU sizes with & without reprogramming.

t =1.00E+08s for VGG11. On the contrary, the (8×4) OU configuration requires reprogramming only twice. However, energy consumption while running inference with such fine-grained OUs is higher than Odin. Odin reduces the OU sizes dynamically, thereby reducing both inference as well as reprogramming energy and latency (reprogramming once). As shown in Fig. 6, Odin reduces the energy by 6.4×, 4×, 1.4×, 3× compared to (16×16), (16×4), (9×8), and (8×4) OU configurations respectively. Similarly, Odin reduces the latency by up to 7.5× compared to homogeneous OUs.

Fig. 7 shows the comparison in predictive accuracy for various homogeneous and Odin-based OU configurations while inferencing using VGG11 DNN model. Without device reprogramming, the predictive accuracy of (16×16) configuration drops by 22% compared to ideal inference accuracy. Homogeneous OUs (16×16) and (8×4) improve predictive accuracy with frequent reprogramming but consume higher total energy and latency compared to Odin as shown in Fig. 6. Thus, Odin achieves higher performance and energy-efficiency without compromising predictive accuracy.

Overall performance evaluation: Since energy-delay-product (EDP) captures both energy and latency in one parameter, we use it as a relevant metric hereafter to evaluate the performance of Odin. Fig. 8 compares the EDP savings of Odin with the existing start-of-the-art homogeneous OU configurations for all the DNN workloads considered in this work. As shown in Fig. 8, Odin reduces the EDP by 3.9×, 2.5×, 1.5×, and 1.9× on average compared to (16×16), (16×4), (9×8), and (8×4) OU configurations respectively across a wide range of image classification datasets. Homogeneous OUs smaller than (16×16) generally result in higher inferencing EDP, as evident from Fig. 8. However, the total EDP is highest in (16×16) due to relatively higher reprogramming overhead. Odin minimizes the inferencing EDP and reprogramming overhead by using varying layer-wise OU sizes via low-overhead OU configuration prediction and online learning.

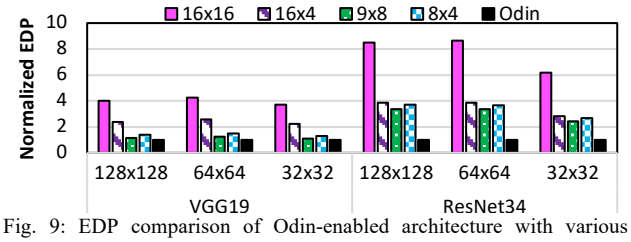


Fig. 9: EDP comparison of Odin-enabled architecture with various homogeneous OUs as the crossbar size varies (normalized to Odin).

D. *Sensitivity Analysis*

We evaluate the performance of the proposed Odin-enabled architecture with various crossbar sizes: 128×128, 64×64, and 32×32. Fig. 9 shows that Odin reduces the EDP by up to 8.5×, 8.7×, and 6.2× over homogeneous OU-based counterparts using ResNet34 (CIFAR-100) on crossbar sizes 128×128, 64×64, and 32×32 respectively. As we scale down the crossbar size, the impact of crossbar non-idealities reduces, reducing the need for reprogramming as well. Nevertheless, Odin outperforms homogeneous OU-based counterparts across varying ReRAM crossbar sizes.

E. *Overhead Analysis*

Next, we quantify the overhead of online learning and layer-wise OU-based computation. OU and ADC controllers require registers, mux, and comparators, introducing an area overhead of 0.005 mm^2 (1.8% of the ReRAM tile area of 0.28 mm^2). OU size prediction consumes 0.14 mW power and introduces 0.9% latency penalty compared to inferencing with *static*, homogeneous (16×16) OU configuration. OU policy update (trained for 100 epochs) introduces energy overhead of 0.22 μJ amortized over multiple inference runs from t_o to 1.00E+08s. Overall, hardware associated with online learning has an area overhead of 0.076 mm^2 (0.2% of the 36-PE system).

VI. Conclusion

Operation unit (OU)-based processing-in-memory (PIM) architectures enable energy-efficient DNN inferencing by exploiting various features of the DNN models. However, each DNN neural layer differs in terms of sparsity, kernel size, etc. Learning to optimize layer-wise OU configuration of an unseen DNN at runtime is a challenging task. In this paper, we propose a low-overhead online learning framework that adapts to unseen DNNs and predicts optimized layer-wise varying OU sizes to reduce energy-delay-product (EDP) with negligible loss in predictive accuracy. The proposed online learning-enabled PIM architecture reduces EDP by up to 8.7 × over state-of-the-art homogeneous OU configurations without compromising predictive accuracy.

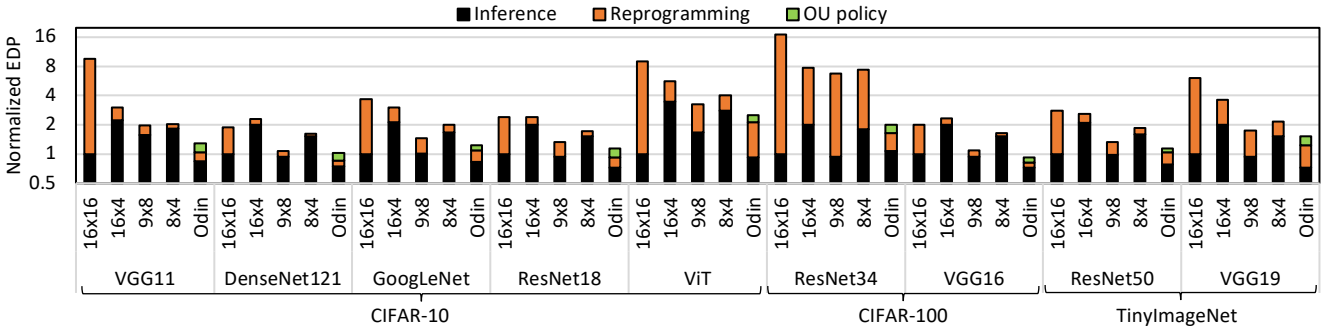


Fig. 8: EDP comparison of Odin-enabled PIM accelerator with state-of-the-art PIM accelerators utilizing *static,* homogeneous OUs while inferencing with various DNNs using CIFAR-10, CIFAR-100, and TinyImageNet datasets. (Normalized with respect to inferencing EDP of (16×16) OU configuration)

REFERENCES

[1] W. Liu et al., "A survey of deep neural network architectures and their applications," *Neurocomputing,* vol. 234, pp. 11-26, 2017.

[2] D. Saraswat et al., "Explainable AI for healthcare 5.0: opportunities and challenges.," *IEEE Access,* vol. 10, pp. 84486-84517, 2022.

[3] H. Zhao, J. Jia and V. Koltun, "Exploring self-attention for image recognition.," *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition,* pp. 10076-10085, 2020.

[4] I. Lauriola, A. Lavelli and F. Aiolli, "An introduction to deep learning in natural language processing: Models, techniques, and tools.," *Neurocomputing,* vol. 470, pp. 443-456, 2022.

[5] K. Roy, I. Chakraborty, M. Ali, A. Ankit and A. Agrawal, "In-Memory Computing in Emerging Memory Technologies for Machine Learning: An Overview," *In 2020 57th ACM/IEEE Design Automation Conference (DAC),* pp. 1-6, 2020.

[6] S. Spetalnick and A. Raychowdhury, "A Practical Design-Space Analysis of Compute-in-Memory With SRAM," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 69, 2022.

[7] T. Soliman et al., "First demonstration of in-memory computing crossbar using multi-level Cell FeFET," *Nature Communications,* vol. 14, no. 1, p. 6348, 2023.

[8] A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars.," *ACM SIGARCH Computer Architecture News,* vol. 44(3), pp. 14-26, 2016.

[9] L. Song, X. Qian, L. Hai and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," *In 2017 IEEE international symposium on high performance computer architecture (HPCA),* pp. 541-552, 2017.

[10] Y.-H. Lin et al., "Performance Impacts of Analog ReRAM Non-ideality on Neuromorphic Computing.," *IEEE Transactions on Electron Devices,* vol. 66, no. 3, pp. 1289-1295, 2019.

[11] C. Huang et al., "Efficient and optimized methods for alleviating the impacts of IR-drop and fault in RRAM based neural computing systems.," *IEEE Journal of the Electron Devices Society,* pp. 645-652, 2021.

[12] I. Chakraborty et al., "Geniex: A generalized approach to emulating non-ideality in memristive xbars using neural networks.," *In 2020 57th ACM/IEEE Design Automation Conference (DAC),* pp. 1-6, 2020.

[13] S. Lee et al., "Learning to Predict IR Drop with Effective Training for ReRAM-based Neural Network Hardware.," *In 2020 57th ACM/IEEE Design Automation Conference (DAC),* pp. 1-6, 2020.

[14] T.-H. Yang et al., "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks.," *In Proceedings of the 46th International Symposium on Computer Architecture,* pp. 236-249, 2019.

[15] C.-Y. Wang, Y.-W. Chang and Y.-H. Chang, "SGIRR: Sparse Graph Index Remapping for ReRAM Crossbar Operation Unit and Power Optimization.," *In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design,* pp. 1-7, 2022.

[16] H. Shin et al., "Effective zero compression on ReRAM-based sparse dnn accelerators.," *In Proceedings of the 59th ACM/IEEE Design Automation Conference,* pp. 949-954, 2022.

[17] A. Bhattacharjee, A. Moitra and P. Panda, "HyDe: A Hybrid PCM/FeFET/SRAM Device-Search for Optimizing Area and Energy-Efficiencies in Analog IMC Platforms," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* vol. 13, 2023.

[18] H. Shin, M. Kang and L.-S. Kim, "Re2fresh: A Framework for Mitigating Read Disturbance in ReRAM-Based DNN Accelerators.," *In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design,* pp. 1-9, 2022.

[19] Y. Park et al., "Unlocking wordline-level parallelism for fast inference on rram-based dnn accelerator.," *In Proceedings of the 39th International Conference on Computer-Aided Design,* 2020.

[20] G. Pedretti, E. Ambrosi and D. Ielmini, "Conductance variations and their impact on the precision of in-memory computing with resistive switching memory (RRAM).," *In 2021 IEEE International Reliability Physics Symposium (IRPS),* pp. 1-8, 2021.

[21] S. Huai et al., "CRIMP: C ompact & R eliable DNN Inference on I n-M emory P rocessing via Crossbar-Aligned Compression and Non-ideality Adaptation.," *ACM Transactions on Embedded Computing Systems ,* Vols. 22, no. 5s, pp. 1-25, 2023.

[22] X. Wu et al., "Block-Wise Mixed-Precision Quantization: Enabling High Efficiency for Practical ReRAM-based DNN Accelerators.," *arXiv preprint arXiv:2310.12182,* 2023.

[23] Z. Song et al., "ReRAM-sharing: Fine-grained weight sharing for ReRAM-based deep neural network accelerator.," *In 2021 IEEE International Symposium on Circuits and Systems (ISCAS),* pp. 1-5, 2021.

[24] Y. Zhang et al., "Pattpim: A practical reram-based dnn accelerator by reusing weight pattern repetitions.," *In 2020 57th ACM/IEEE Design Automation Conference (DAC),* pp. 1-6, 2020.

[25] C.-Y. Tsai et al., "RePIM: Joint exploitation of activation and weight repetitions for in-ReRAM DNN acceleration.," *In 2021 58th ACM/IEEE Design Automation Conference (DAC),* pp. 589-594, 2021.

[26] G. Narang, C. Ogbogu, J. Doppa and P. Pande, "TEFLON: Thermally Efficient Dataflow-Aware 3D NoC for Accelerating CNN Inferencing on Manycore PIM Architectures.," *ACM Transactions on Embedded Computing Systems,* 2024.

[27] G. Narang et al., "Uncertainty-Aware Online Learning for Dynamic Power Management in Large Manycore Systems.," *In 2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED),* 2023.

[28] D. Xu et al., "Survey on multi-output learning.," *IEEE transactions on neural networks and learning systems,* vol. 31(7), pp. 2409-2429, 2019.

[29] C. Ogbogu et al., "Energy-Efficient ReRAM-Based ML Training via Mixed Pruning and Reconfigurable ADC.," *In 2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED),* pp. 1-6, 2023.

[30] M. Saberi, R. Lotfi, K. Mafinezhad and W. Serdijn, "Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol. 58(8), pp. 1736-1748, 2011.

[31] H. Jin et al., "ReHy: A ReRAM-based digital/analog hybrid PIM architecture for accelerating CNN training," *IEEE Transactions on Parallel and Distributed Systems,* vol. 33, no. 11, pp. 2872-2884, 2021.

[32] X. Peng et al., "Benchmarking monolithic 3D integration for compute-in-memory accelerators: overcoming ADC bottlenecks and maintaining scalability to 7nm or beyond," *In 2020 IEEE International Electron Devices Meeting (IEDM),* pp. 30-4, 2020.

[33] Z. He, J. Lin, R. Ewetz, J. Yuan and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," *In Proceedings of the 56th Annual Design Automation Conference (DAC),* pp. 1-6, 2019.

[34] W. Chen et al., "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors.," *In 2018 IEEE International Solid-State Circuits Conference-(ISSCC),* pp. 494-496, 2018.