Endurance-Aware Deep Neural Network Real-Time Scheduling on ReRAM Accelerators

Shi Sha
College of Business and Engineering
Wilkes University
Wilkes-Barre, PA, United States
shi.sha@wilkes.edu

Xiaokun Yang
College of Science and Engineering
University of Houston-Clear Lake
Houston, TX, United States
yangxia@uhcl.edu

Trent M. Szczecinski
College of Business and Engineering
Wilkes University
Wilkes-Barre, PA, United States
trent.szczecinski@wilkes.edu

Daniel Whitman
College of Business and Engineering
Wilkes University
Wilkes-Barre, PA, United States
daniel.whitman@wilkes.edu

Wujie Wen
Department of Computer Science
North Carolina State University
Raleigh, NC, United States
wwen2@ncsu.edu

Gang Quan
Electrical and Computer Engineering Department
Florida International University
Miami, FL, United States
gang.quan@fiu.edu

***Abstract*—Achieving accurate multi-modal Deep Neural Networks (DNN) testing often requires operating rich model parameters under limited computing and memory resources. The low-cost Resistive Random-Access Memory (ReRAM)-based DNN accelerator is a promising solution for such an application thanks to its inherent processing-in-memory capability. However, its lifetime, which is crucial for applications with strict reliability standards such as self-driving cars, can be significantly limited due to: 1) the need to frequently switch weights among different models for real-time streaming applications; 2) the low endurance of ReRAM devices compare to DRAMs by orders of magnitude. This work proposed an Endurance-Aware multi-modal DNN Scheduling (EAS) strategy to address this issue using real-time techniques. First, a pre-processing methodology transformed a DNN into an end-to-end execution sequence for partitioning and scheduling. Then, a periodic real-time scheduling method was developed via data reusing for extending ReRAM programming cycles under deadline constraints. The experiment results showed that our EAS approach can extend the baseline ReRAM accelerator's lifetime from 0.98 years to 3.14 years on average at a low scheduling overhead.**

***Index Terms*—Resistive random-access memory (ReRAM), deep neural network (DNN), real-time, reliability, and streaming applications.**

I. Introduction

Multi-Modal Deep Neural Network (DNN) testing with constrained response latency is indispensable for future intelligent systems [1]. For example, autonomous driving needs the fusion of heterogeneous neural network architectures, including CNNs and RNNs for sensor-streamed video, audio, location tracking, and thermal video processing [2]. Moving from computing to a data-centric paradigm, the miniature device parameters–such as cost, area, and energy consumption–impose intra-/inter-task resource contention upon intrinsic processing demands, which could be even more challenging when sustaining hardware endurance upon complicated scheduling policies. Since aging-induced memristor defects downgrade inference accuracy significantly [3], this work focuses on prolonging the Resistive Random-Access Memory (ReRAM) lifetime in the context of distinguished DNN data flow properties using real-time scheduling techniques.

ReRAM-based Processing-In-Memory (PIM) accelerators achieve fast and energy-efficient DNN execution by inherent in-situ multiply-accumulate operations [4], [5], [6], [7], e.g. the ReRAM convolution computing efficiency can achieve $> 10^3$ times over traditional ASIC designs [2]. However, a ReRAM can only sustain a limited number of programming cycles $\approx 10^8$ [8], [9], [10] as opposed to $> 10^{16}$ for DRAM memory [11]. For example, the ReRAM lifetime is 0.95 years for sequential DNN testing [12] on an autonomous vehicle at 40 frames per second (FPS) sensing rate [13] and running 2 hours per day. The need to simultaneously execute multi-modal DNNs with targeted deadlines in such resource-limited PIM accelerators can further aggravate the lifetime issue since weights shall be frequently reprogrammed to execute different models.

Frequently programming ReRAM not only expedites memristors' aging effect but also extends response latency, increases local power density and temperature, and downgrades inference accuracy. Writing on ReRAM cells costs a typical latency of 1 write-verify cycle ($\approx 500ns$) for 1-bit/cell [14], during which the high voltages and large currents are applied to establish the conductive filaments at targeting memristor resistances. Meanwhile, the large conductance and current result in soaring local power density that increases internal and ambient temperature [15]. As memristors' resistivity is strongly temperature-sensitive, the thermal instability leads to inaccurate programming conductance. Consequently, conductance deviation from the pre-trained DNN can downgrade inference accuracy [16].

The ReRAM endurance can be leveraged by reducing the memory writes [17], [18], [19], [20], adjusting programming modes [16], or managing runtime thermal status [15], [9]. However, some solutions exhibited either accuracy degrada-

2769-5654/23/$31.00 ©2023 IEEE
DOI 10.1109/CSCI62032.2023.00072

tion [17], [18], [20] or scalability limitations [19], [9]. To satisfy real-time constraints, some works took advantage of the "intrinsic resilience" of DNNs to approximate results by judiciously reducing the data volume and the response latency [21], [22], [23], e.g. formulating the DNN as a scalable computational model [24], [25] or adjusting the NN hyper-parameters, such as compression [26], network pruning [27], precision scaling [28], etc. However, these solutions still suffered from accuracy degradation. We propose to take advantage of the slack time between the task arrival and its deadline for reusing the mapped data on the ReRAM and testing a maximal number of feature maps sharing the same kernels without shrinking network architectures or any accuracy loss.

This work utilized real-time techniques to reschedule DNN data flow for ReRAM endurance enhancement under deadline constraints, which was orthogonal to other state-of-the-art works. Our proposed methodologies can also be applied to different reprogrammable devices when mitigating the intensive reprogramming-induced lifetime reduction in the real-time domain. The contributions included

- We designed a DNN pre-processing methodology that converted a DNN to an end-to-end execution sequence with bounded layer-wise computational and data storage demands for partitioning and scheduling.
- We developed a real-time DNN inference scheduling framework on ReRAMs that can reuse the mapped kernel weights for minimizing the memristor writes and enhancing the ReRAM endurance under time constraints.
- Our experimental results showed that our approach can extend the baseline ReRAM lifetime by 3.2 times at high feasibility for different deadlines and workload intensities with low scheduling overhead.

The rest of this paper is organized as follows. Section II presents the system architecture, task model, and problem formulation. Section III presents our approach. Section IV shows the experiment results and is followed by a conclusion in Section V.

II. Preliminaries

This section presents the architecture and task models followed by the problem formulation. The **bold** characters represent the vectors and matrices, and non-bold characters are used for ordinary variables and coefficients. All of the matrices/vectors/values are in the real number domain $\mathbb{R}$.

A. Architecture Overview

We adopt the 3D stacking ReRAM [9], [18], [29], [8] as a full-fledged DNN accelerator $\mathcal{R}$, as shown in Figure 1. The ReRAM accelerator consists of N tiles as $\mathcal{R} = \{T_1, \cdots, T_N\}$, which can be independently controlled and configured. The data and signal are transmitted through vertical Through-Silicon-Via (TSV). Each tile, similar to [30], has a mesh of multiply-accumulator (MA) units, each of which consists of arrays of homogeneous crossbars (Xbar). Other components, such as shift-and-add, activation (σ), and pooling units are built on each tile and their execution times are added to

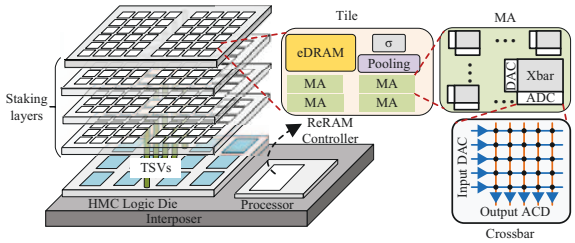

Fig. 1. Architecture Overview

the MA operations for bounding the layer-wise worst-case execution time (WCET). Each tile controls memory access independently and has a fixed size of eDRAMs [18] for storing transient results. Without losing generality, this work assumes that each tile cannot be shared by multiple applications, and all tiles run at a uniform frequency.

DNN tasks use memristors to perform analog computing and their transient results are stored in eDRAMs. Let M and E represent the sizes of memristors and eDRAMs of a tile, respectively. The total sizes of memristors and eDRAMs on the ReRAM are $N \cdot M$ and $N \cdot E$, respectively. Assume n_k tiles are assigned to the k-th task $\Gamma_k \in \mathbf{\Gamma}$, the allocated sizes of memristors and eDRAMs are $n_k \cdot M$ and $n_k \cdot E$, respectively.

B. Task Model

Consider that a DNN inference task set $\mathbf{\Gamma}$ contains θ independent tasks of different models as $\mathbf{\Gamma} = \{\Gamma_1, \cdots, \Gamma_\theta\}$. The k-th task Γ_k has L_k layers as $1 \leq l \leq L_k$, contains s_k real-time streaming instances sharing the same kernel set, and is constrained by an end-to-end deadline equal to the period of D. For Γ_k, each layer abstracts the characteristic elements by strikingly convolute a set of kernels with data from the previous layer (layer $l-1$) to generate feature maps of the next layer (layer l) [31], where the output feature map of layer $l-1$, denoted as $\mathbf{o}_k^{l-1}$, is temporarily stored in eDRAMs and serves as the input feature map of layer l. Let Γ_k^l be the l-th layer of Γ_k. Then, the j-th computational instance of Γ_k^l can be denoted as $y_{k,j}^l = \mathbf{o}_{k,j}^{l-1^\top} \mathbf{W}_k^l + b_k^l$, where $y_{k,j}^l$ is the j-th convoluted output of Γ_k^l, $\mathbf{W}_k^l \in \mathbb{R}^{m^{l-1} \times m^l}$ contains the weight parameters and $b_k^l \in \mathbb{R}^{m^l}$ is the bias [32]. Next, an activation function, e.g. sigmoid $\sigma(\cdot)$, rectifies the convoluted instance as $o_{k,j}^l = \sigma(y_{k,j}^l)$, where $o_{k,j}^l$ is the j-th instance's output neuron value of Γ_k^l. The latter layer can be executed only after the completion of the previous one due to data dependency [33].

To quantify the inferencing resource demands, assuming that executing Γ_k^l needs a size of m_k^l memristors to load weights and a size of e_k^l eDRAMs to store intermediate feature maps. Then, in the run time, if loading d consecutive layers' kernel weights to its partition at the same time, the total size of weights $\sum_{l=l1}^{l1+d} m_k^l$ should not exceed $n_k \cdot M$. Meanwhile, if a collection of intermediate feature maps, e.g. $\mathbf{o}_k^l \in \mathbb{J}$, need to be stored in the eDRAMs, the total size of $\sum_{\mathbf{o}_k^l \in \mathbb{J}} e_k^l$, should not exceed $n_k \cdot E$. Since convolutional

operations dominate the DNN computation [34], we let τ_k^l be the WCET of Γ_k^l, including convolution, pooling, activation, and their communications through buses.

C. Problem Formulation

DNN inference involves a combination of convolutional, normalization, pooling, and classification operations [35]. Since the number of weight parameters is magnitudes larger than neurons on most DNNs [32], this work focuses on reusing the mapped weights to reduce the memristor writes and improve ReRAM endurance. When a ReRAM is large enough to load all the kernels in $\mathbf{\Gamma}$, i.e. $N \cdot M \geq \sum_{k=1}^{\theta} \sum_{l=1}^{L_k} m_k^l$, memristors only need to be programmed once. However, when multiple concurrent DNNs share one ReRAM under deadline constraints, it needs to update the weights frequently as network inference proceeds. Therefore, the short programming cycles, which are detrimental to hardware endurance, motivate us to develop innovative multi-modal DNN scheduling methodologies to maximally attain endurance under the targeted deadline. With the above analyses, the problem to be solved in this work is formulated as follows.

Problem 1: Given a real-time streaming DNN inference task set $\mathbf{\Gamma} = \{\Gamma_1 \cdots, \Gamma_\theta\}$, with the end-to-end deadline of D, to be executed on a ReRAM platform $\mathcal{R} = \{T_1, \cdots, T_N\}$, the problem is to fit $\mathbf{\Gamma}$ into $\mathcal{R}$ and create real-time schedules to maximally attain the ReRAM endurance under the deadline constraint.

III. Our Approach

With the formulated problem, we present our ReRAM partitioning, DNN pre-processing, and real-time scheduling approaches for endurance enhancement. The main idea is to convert heterogeneous DNN layers into a sequence of homogeneous sub-layers to fit into partitioned resources and reuse the mapped kernel weights in consecutive layers to prolong the ReRAM endurance without violating deadline constraints.

A. ReRAM Partitioning

For a multi-modal DNN task set, the number of partitioned tiles for each task is proportional to the workload intensity. Assuming a uniform sensing rate is applied to the multi-modal DNN tests, e.g. in-sync video and audio for autonomous driving, increasing s_k leads to a longer response latency, as explained later in Figure (4b), so more tiles should be allocated to Γ_k under deadline constraints as $n_k = \lfloor N \cdot \frac{s_k \cdot \sum_{l=1}^{L_k} m_k^l}{\sum_{k=1}^{\theta} \sum_{l=1}^{L_k} s_k \cdot m_k^l} \rfloor$ in the run time. However, the sizes of parameters and feature maps for different layers vary significantly as shown in Figure 2, which may exceed the allocated tiles' total capacity or lead to an unbalanced execution pipeline. To this end, we first develop a DNN pre-processing method to decompose "large" layers into a sequence of end-to-end sub-layers for fitting each DNN into partitioned resources. Then, an endurance-aware real-time scheduling approach is introduced.

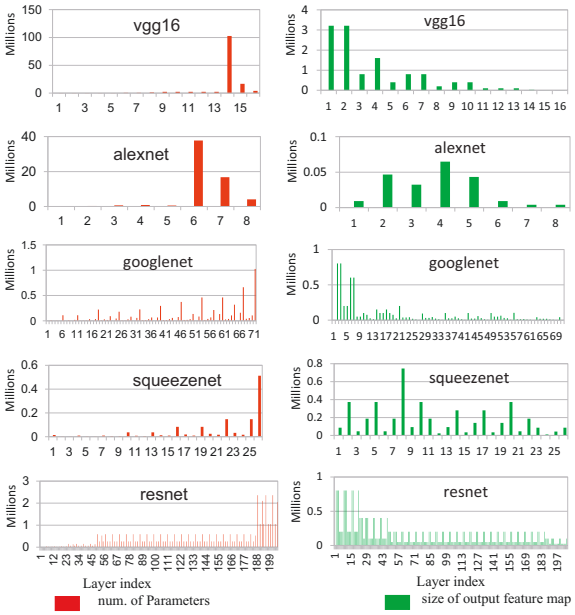

Fig. 2. Layer-wise parameter and feature map sizes vary significantly before processing

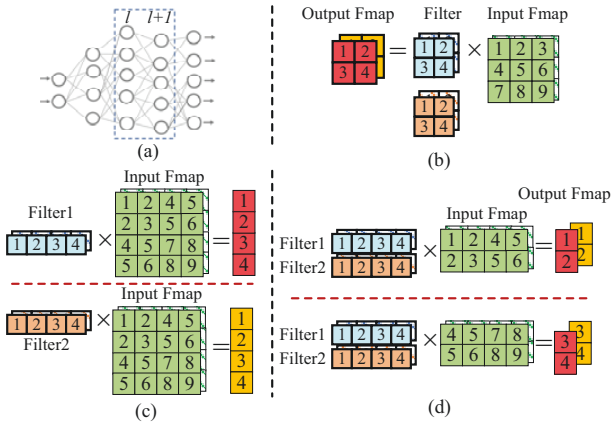

Fig. 3. Layer decomposition: (a) Given DNN workload; (b) A layer before decomposition; (c) Split filters into multiple sub-layers; (d) Split the feature map inputs into multiple sub-layers.

B. DNN Pre-processing

When allocating a DNN to its partitioned tiles, the heterogeneity of the layer-wise parameters may exceed the allocated resources. In the meantime, different volumes of multiply-accumulation operations cause varying layer-wise latency that can be detrimental to throughput performance due to the unbalanced execution pipeline. To resolve the layer heterogeneity-induced resource contention and unbalanced pipeline for a better response latency, we propose a pre-processing stage for bounding the sizes of parameters, the input feature maps, and the WCET for each layer. Then, each DNN inference task can be treated as a sequence of homogeneous layers for

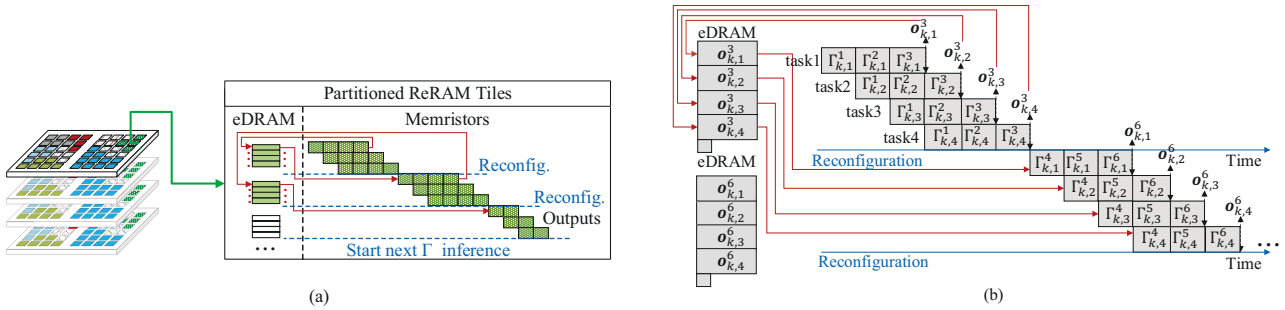

Fig. 4. Endurance-aware real-time scheduling. Notations $\Gamma^l_{k,\psi}$ and $\mathbf{o}^l_{k,\psi}$ represents the ψ-th inference instance of Γ^l_k and its output feature map, respectively. (a) ReRAM accelerator partition. (b) Data flow of multi-DNN endurance-aware pipelined execution.

scheduling.

To put the DNN pre-processing into perspective, given an arbitrary network as Figure 3(a), the kernel filters convolute through the input feature maps on layer l to output feature maps for layer $l+1$ as Figure 3(b). Assume layer l is pending to be decomposed, let $\widehat{m_k}$ be the layer-wise parameter size bound and $\widehat{f_k}$ be the feature map size bound for Γ_k. We first split kernels according to $\widehat{m_k}$ without reshaping the input feature maps as Figure 3(c). Then, we split its input feature maps according to $\widehat{f_k}$ into multiple sub-layers as Figure 3(d). Note that splitting kernels or input feature maps also impacts the output feature map sizes. The post-processing DNN should contain all consecutive layers with bounded kernel and feature map sizes, which caps the total multiply-accumulation operations. Since the multiply-accumulation dominates and is proportional to the WCET, the post-processing layer-wise latency is also bounded as $\widehat{\tau_k}$. Overall, the pre-processing stage could dramatically enhance DNN's real-time predictability and schedulability.

To enhance the ReRAM endurance, we propose to reuse mapped kernel weights to test as many feature maps as possible under deadline constraints. In addition, the tasks sharing the same kernels are expected to be executed in the *pipeline* to minimize the response latency [33], [36]. When the partitioned tiles cannot hold all kernels at once, we define *reconfiguration* as a renewal of kernel weights on the memristors as DNN inference proceeds. Then, each *configuration* can only hold the kernels of several consecutive layers, if not all. Therefore, an inference task usually needs multiple reconfigurations as shown in Figure 4(a). In addition, due to the continuity of DNN execution, once the inference starts, it must proceed until the completion of all layers in consecutive configurations. Otherwise, storing intermediate results consumes eDRAM capacity, and squeezes later configurations' data-preserving space.

Specifically, as shown in Figure 4(b), the kernels that belong to one configuration can be reused by multiple inferencing instances, and partitioned eDRAMs are shared among them. For each inferencing instance, the previous layer's output feature map is stored in the eDRAM and immediately transferred to the next layer, except that the last output feature map of a configuration should be stored in the eDRAM longer to serve as the first input feature map for the following configuration. Since the feature map sizes are bounded, as $\widehat{f_k}$ for task Γ_k, and the number of intermediate feature maps to be stored in the eDRAM are identical for all configurations, eDRAM overflow should never occur.

C. Endurance-Aware Real-Time Scheduling

For each task $\Gamma_k \in \mathbf{\Gamma}$, we are ready to create its real-time schedule for maximizing the programming cycles (C_k) and enhancing the ReRAM endurance under the hardware and timing constraints. Let v_k be the number of instances reusing the same set of kernels when executing Γ_k, where v_k is a positive integer ($v_k \in \mathbb{Z}^+$). The larger the v_k, the more input feature maps can reuse the same mapped kernel weights, and, thus, ReRAM has longer programming cycles and better endurance. However, from Figure 4(b), we can see that increasing v_k leads to a longer inference latency and a higher eDRAM utilization in each configuration. Let v_k^D and v_k^E be the maximal v_k values concerning the deadline and the assigned eDRAM size, respectively. Then, we have

$$v_k = min\{\ v_k^D,\ \ v_k^E,\ \ s_k\ \},\quad \text{where}\ \ v_k^E = \lfloor n_k \cdot E/\widehat{f_k} \rfloor. \quad (1)$$

To identify the maximal v_k^D, let d_k be the execution pipeline depth. The value of d_k is constrained by the largest number of layers that can be concurrently loaded onto the partitioned tiles in one configuration as $d_k = \lfloor n_k \cdot M/\widehat{m_k} \rfloor$. Assume that the completion of Γ_k needs r_k configurations, we have $r_k = \lceil L_k/d_k \rceil$. If the last configuration contains d_k^{last} remnant layers, we can infer $d_k^{last} = L_k - (r_k - 1)d_k$. To guarantee the completion of Γ_k within the time frame $[0, D]$, v_k^D needs to satisfy

$$C_k(r_k - 1) + \widehat{\tau_k} \cdot (v_k^D + d_k^{last} - 1) \le D,$$
$$\text{where}\ \ C_k = \widehat{\tau_k} \cdot (v_k^D + d_k - 1). \quad (2)$$

Solving Equation (2), we have

$$v_k^D = \lfloor \frac{D}{r_k \cdot \widehat{\tau_k}} + \frac{d_k - d_k^{last}}{r_k} - d_k + 1 \rfloor. \quad (3)$$

Algorithm 1 Endurance-aware DNN scheduling (EAS)

1: **Inputs:** Task set $\mathbf{\Gamma}$, ReRAM platform $\mathcal{R}$, Deadline D
2: **Outputs:** $\mathbb{S}(v_k, C_k, d_k, r_k)$, where $1 \leq k \leq \theta$
3: Partition $\mathcal{R}$ and identify n_k
4: Pre-processing initialization: $\mathbf{\Gamma'} = [\ ]$, $k = 1$
5: **while** $k \leq \theta$ **do**
6: linearly search $max(d_k)$ and determine $\widehat{m_k}$ and $\widehat{f_k}$
7: **while** $\exists\ sizeof\ (\Gamma_k^l) > \widehat{m_k}$ or $\widehat{f_k}$ **do**
8: Decompose Γ_k^l kernels or input feature maps
9: Update Γ_k
10: **end while**
11: $\mathbf{\Gamma}^{'} += \Gamma_k$, $\mathbf{\Gamma} -= \Gamma_k$, $k++$
12: **end while**
13: Scheduling initialization: $\mathbb{S} = [\ ]$, $k = 1$
14: **while** $\mathbf{\Gamma} \neq [\ ]$ **do**
15: Determine d_k, r_k and d_k^{last}
16: Identify v_k and C_k in Equation (1) and (2)
17: **if** $v_k < s_k$ **then**
18: Return Infeasible
19: **end if**
20: $\mathbb{S} += \mathbb{S}_k$, $\mathbf{\Gamma} -= \Gamma_k$
21: **end while**
22: Output $\mathbb{S}(v_k, C_k, d_k, r_k)$

The pre-processing procedures and endurance-aware real-time scheduling approaches are summarized in Algorithm 1.

In Algorithm 1, the larger the d_k, the less r_k and the higher the endurance. Thus, assuming $v_k^D = s_k$ in Equation (2), we linearly search the maximum d_k without exceeding the partitioned eDRAMs capacity or violating deadline D for determining $\widehat{m_k}$ and $\widehat{f_k}$ in line 6. With the bounded memristors and feature maps, Algorithm 1 lines 4-12 can be applied to decompose kernels and feature map inputs. Next, we can readily create endurance-aware real-time DNN inference schedules for each $\mathbb{S}_k(v_k, C_k, d_k, r_k)$ as shown in lines 13-21. The effectiveness of Algorithm 1 are formalized and proved in Lemma 3.1 and 3.2, assuming that the post-processing DNNs have their layer-wise size of weights and size of feature maps ideally equal to their corresponding bounds.

Lemma 3.1: **[Temporal Tightness]** When $s_k \to \infty$ and $n_k \cdot E \to \infty$, parameter v_k is tight regarding deadline D.

Proof Since $n_k \cdot E \to \infty$, we have $v_k^E \to \infty$, and, thus, $v_k = v_k^D$ in Equation (1). Assume $\tilde{v}_k^D = v_k^D + 1$ satisfies Equation (2), where $\tilde{v}_k^D, v_k^D \in \mathbb{Z}^+$, taking $\tilde{v}_k^D$ into Equation (2), we have $\tilde{v}_k^D \leq Q$ and $Q = \frac{D}{r_k \cdot \widehat{\tau}_k} + \frac{d_k - d_k^{last}}{r_k} - d_k$. Since $v_k^D = \lfloor Q + 1 \rfloor$ in Equation (3), we have $v_k^D \geq Q$. Therefore, $\tilde{v}_k^D \leq Q \leq v_k^D$ is contradict with the initial assumption of $\tilde{v}_k^D = v_k^D + 1$.

Lemma 3.2: **[Spatial Tightness]** When $D \to \infty$ and $s_k \to \infty$, eDRAM can be filled with intermediate feature maps as $v_k \cdot \widehat{f_k} = n_k \cdot E - \epsilon$, where ϵ is a very small value.

Proof When $D \to \infty$, we have $v_k^D \to \infty$ from Equation (3). In addition, since $s_k \to \infty$, from Equation (1), we have $v_k = \lfloor n_k \cdot E / \widehat{f_k} \rfloor$ and v_k is the largest integer that satisfies $v_k \cdot \widehat{f_k} \leq n_k \cdot E$.

TABLE I
RERAM ACCELERATOR PARAMETERS [30]

Component	Spec	Value
ReRAM	N_{TL}	192 tiles
MA	number	12 per tile
Xbar	number	8 per MA
	size	128 × 128 cells
	bit	2 per cell
eDRAM	num. of banks	2 per tile
	size	64 KB

With Lemma 3.1 and 3.2, we can infer that Algorithm 1 can maximally explore the temporal slack time and spatial eDRAM storage when reusing the mapped kernels on the ReRAM, either constrained by the given real-time deadline or by the partitioned eDRAM storage size, when s_k is not a limiting factor.

IV. EXPERIMENTAL RESULTS

In Section IV, we verified our design and compared their endurance and real-time performances with state-of-the-art techniques.

A. Experimental Setup

We adopted the similar ReRAM accelerator settings as [30] in Table I. All DNN inferences employed 16-bit operation if not otherwise specified. The DNN inference task set was randomly composed of VGG16, AlexNet, GoogleNet, SqueezeNet, and ResNet, which had different kernel sizes, output shapes, and parameter volumes [12]. The execution time of each layer was normalized to the system ticks. We assumed an optimized and negligible memristor writing overhead [16], [31] compared with the longer inference latency in the experiment. The simulation was written in C++ and ran at Intel-i9 CPU at 2.40 GHz. We evaluated the effectiveness of our proposed approach by comparing the following methods.

- Baseline Approach (**BL**): The single-priority sequential execution as the default setting in the state-of-art DNN frameworks, such as Caffe, TensorFlow, and Torch [12].
- Pipelined Approach (**PA**): The best-effort pipelined DNN execution to maximize the system throughput [33], [36].
- Endurance-aware DNN scheduling Approach (**EAS**): Our proposed endurance-aware real-time multi-modal DNN scheduling in Algorithm 1.

The experiment was conducted by setting the deadlines D from 30 to 240ms with an increment of 30ms. Let $Ub(s_k)$ be the upper boundary of s_k and serve as an index for the workload intensity given fixed DNN models. For each D setting, we increased $Ub(s_k)$ from 2 to 24 at the step size of 2. Then, for each pair of D and $Ub(s_k)$, the composition of the task set $\mathbf{\Gamma}$, in terms of Γ_k and s_k, were generated for $1K$ random cases and took their average results.

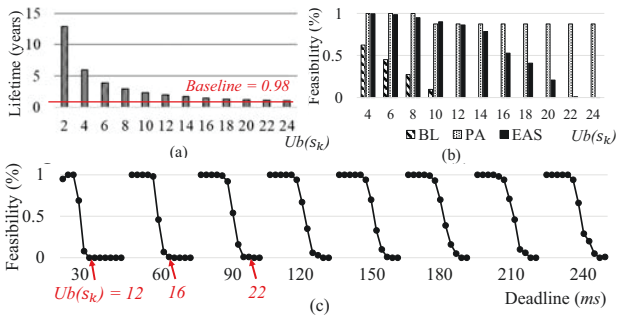

Fig. 5. (a) ReRAM endurance; (b) Feasibility comparisons; (c) Feasibility changes for different D and $Ub(s_k)$ of the EAS approach.

B. Endurance and Feasibility Comparisons

We adopted the typical ReRAM endurance as 4.14×10^8 writes [8], [9], [10]. Assuming a ReRAM in a commercial autonomous vehicle runs 8 hours/day at 40 frames per second [13] sensing rate, it can sustain 0.98 years using the **BL** approach. Although **PA**'s pipelined execution pattern benefits its response time, it does not consider saving memristor writes for endurance attainment and has the same endurance as **BL**.

The proposed **EAS**'s lifetime monotonically decreases as the total amount of workload increases. As shown in Figure 5(a), when $Ub(s_k)$ increased from 2 to 24, the average ReRAM lifetime reduced from 12.89 to 0.98 years in an average of 3.14 years. Overall, the proposed **EAS** can attain more than $3.2\times$ lifetime compared with the baseline.

Figure 5(b) compares the feasibility of different approaches. The **BL** approach has the lowest feasibility because it executes DNNs in sequential patterns without pipelining or overlapping different tasks, which results in the longest response time. For example, as $Ub(s_k)$ increased from 4 to 10, the feasibility of **BL** decreased from 62.4% to 9.7% and became totally infeasible when $Ub(s_k) \geq 12$. On the contrary, the **PA** approach pipelines different layers to achieve the best-effort throughput, so the feasibility of **PA** is the highest among these methods, which exceeded 87.5% for all the cases.

Our proposed **EAS** approach feasibility is better than **BL**, but lower than **PA**. The reason is that **EAS** adopts the pipelined execution pattern; however, due to endurance consideration, when reusing the mapped weights, the early finished layers must wait until the completion of other tasks that share the same kernel. Therefore, the **EAS** response time for each task becomes longer than **PA**. As shown in Figure 5(b), when $Ub(s_k)$ increased from 4 to 22, the feasibility of **EAS** decreased from 99.6% to 10.1% with an average of 60.3%.

Figure 5(c) evaluates the feasibility of our proposed **EAS** approach for different D and $Ub(s_k)$. For each deadline, the feasibility decreases as the amount of workload increases. Further, as the deadline becomes larger, it could achieve better feasibility since more tasks can meet their deadlines. For example, when D increased from 30 to $90ms$, the earliest all-infeasible $Ub(s_k)$ extended from 12 to 22 and kept at 22 when $D \geq 90ms$, which indicated the feasibility improvement as the deadline was relaxed.

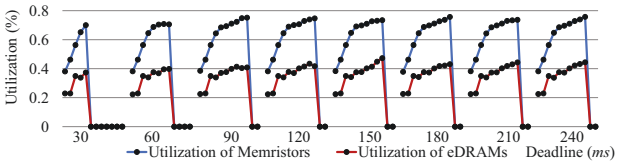

Fig. 6. Memristor and eDRAM utilization comparisons for different D and $Ub(s_k)$ of the EAS approach.

C. System Utilization

This section evaluates the system utilization on memristors and eDRAMs for the proposed **EAS** approach. The *utilization of memristors* is defined as the number of engaged memristors versus the total ReRAM memristors within one period. Similarly, the *utilization of eDRAMs* is defined as the size of eDRAMs storing intermediate results versus the system-wide eDRAM capacity in one period.

Figure 6 shows that both memristors' and eDRAMs' utilization increase as the total amount of workload become larger for all different deadline settings. For example, when $D = 120ms$ and $Ub(s_k) = 2$, the memristor and eDRAM utilization was 38.1% and 22.4%, respectively. As $Ub(s_k)$ increased to 20, the memristor and eDRAM utilization became 74.7% and 41.7%, respectively. The reason is that when there are fewer inferencing instances for a task, the completion time may be earlier than the deadline, so both memristors' and eDRAMs' utilization could be low because of idle time. However, as $Ub(s_k)$ increased to the extent that the completion time of $\mathbf{\Gamma}$ violated the deadline, the infeasible cases' utilization for memristors and eDRAMs was set as 0.

We also observed that for all different deadline settings, the memristor utilization was higher than the eDRAM utilization for feasible cases. The reason is that the number of weights in most DNNs is several magnitudes larger than neurons. Therefore, in Figure 6, the maximum memristor utilization can reach 75.7% at $D = 240ms$, but the maximum eDRAM utilization can only achieve 47.2% at $D = 150ms$. The "headroom" of the memristor utilization was caused by the margin between the actual size of weights m_k^l and the bounded size of weights $\widehat{m_k}$ per layer.

D. Execution Overhead

Lastly, we evaluated the execution overhead of our proposed **EAS** method. We found that the partitioning and scheduling consumed a negligible overhead, e.g. the average partitioning and scheduling overheads were $0.29\mu s$ and $5.58\mu s$, respectively. A majority of the time was spent on decomposition at 0.45*ms* on average. As shown in Table II, the CPU time increases as $Ub(s_k)$ becomes larger. The reason is that the larger number of work items that can share the same kernel, the lower bounded layer-wise weights, and feature map sizes may apply, and, thus, layer decomposition is needed

TABLE II
CPU TIME OF EAS FOR DIFFERENT WORKLOAD INTENSITIES. (IN ms)

$Ub(s_k)$	4	8	12	16	20	24	Avg
CPU time	0.23	0.34	0.39	0.43	0.61	0.71	0.45

to generate more sub-layers in pre-processing. Overall, our proposed **EAS** method is computationally efficient.

V. CONCLUSION

ReRAM exhibits high computing and energy efficiency, but the low endurance limits its lifetime and reliability. When multi-modal DNNs share limited crossbars under constrained latency, the high programming density exacerbates the ReRAM lifetime. In this work, we developed a novel endurance-enhancement real-time multi-DNN scheduling approach, including resource partitioning, DNN pre-processing, and real-time scheduling. The proposed approach can maximally explore the temporal and spatial ReRAM resources for reusing the mapped kernels and extending programming cycles, which dramatically leverages ReRAM endurance without any accuracy loss. The experimental results showed that our proposed method can effectively and efficiently enhance the ReRAM lifetime by $3.2\times$ compared to the state-of-the-art techniques.

ACKNOWLEDGEMENT

This work was supported by the Provost's Research and Scholarship Fund at Wilkes University, the Pennsylvania State Space Grant Consortium (PSGC), NSF CCF-2011236 and NSF CCF-2006748.

REFERENCES

[1] B. Li, Y. Wang, and Y. Chen, "Hitm: High-throughput reram-based pim for multi-modal neural networks," in *ICCAD*, 2020.
[2] Q. Lou, W. Wen, and L. Jiang, "3dict: A reliable and qos capable mobile process-in-memory architecture for lookup-based cnns in 3d xpoint rerams," in *ICCAD*, 2018.
[3] L. Xia, Mengyun Liu, Xuefei Ning, K. Chakrabarty, and Yu Wang, "Fault-tolerant training with online fault detection for rram-based neural computing systems," in *DAC*, 2017.
[4] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *SIGARCH Comput. Archit. News*, 2014.
[5] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, 2016.
[6] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014.
[7] Ming Cheng, Lixue Xia, Zhenhua Zhu, Yi Cai, Yuan Xie, Yu Wang, and Huazhong Yang, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *DAC*, 2017.
[8] M. Valad Beigi and G. Memik, "Thor: Thermal-aware optimizations for extending reram lifetime," in *IPDPS*, 2018.
[9] M. Zhou, M. Imani, S. Gupta, and T. Rosing, "Thermal-aware design and management for search-based in-memory acceleration," in *DAC*, 2019.
[10] Y. Fan, P.-T. Liu, C.-C. Chen, and C.-C. Chang, "High endurance and multilevel operation in oxide semiconductor-based resistive ram using thin-film transistor as a selector," *ECS Solid State Letters*, 2015.
[11] Y. Cai, Y. Lin, L. Xia, X. Chen, S. Han, Y. Wang, and H. Yang, "Long live time: Improving lifetime and security for nvm-based training-in-memory systems," *TCAD*, 2020.
[12] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, 2017.
[13] J. Kocic, N. Jovicic, and V. Drndarevic, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," *Sensors*, 2019.
[14] L. Gao, P.-Y. Chen, and S. Yu, "Programming protocol optimization for analog weight tuning in resistive memories," *IEEE Electron Device Letters*, 2015.
[15] S. Zhang, G. L. Zhang, B. Li, H. H. Li, and U. Schlichtmann, "Lifetime enhancement for rram-based computing-in-memory engine considering aging and thermal effects," in *AICAS*, 2020.
[16] W. Wen, Y. Zhang, and J. Yang, "Renew: Enhancing lifetime for reram crossbar based neural network accelerators," in *ICCD*, 2019.
[17] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang, "Switched by input: Power efficient structure for rram-based convolutional neural network," in *DAC*, 2016.
[18] X. Liu, M. Zhou, T. S. Rosing, and J. Zhao, "Hr3am: A heat resilient design for rram-based neuromorphic computing," in *ISLPED*, 2019.
[19] Y. Zhang, D. Feng, J. Liu, W. Tong, B. Wu, and C. Fang, "A novel reram-based main memory structure for optimizing access latency and reliability," in *DAC*, 2017.
[20] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on rram," in *ASP-DAC*, 2017.
[21] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *DAC*, 2013.
[22] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "Rram-based analog approximate computing," *TCAD*, 2015.
[23] P. Panda, A. Sengupta, S. S. Sarwar, G. Srinivasan, S. Venkataramani, A. Raghunathan, and K. Roy, "Invited — cross-layer approximations for neuromorphic computing: From devices to circuits and systems," in *DAC*, 2016.
[24] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *RTSS*, 2018.
[25] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher, "Scheduling real-time deep learning services as imprecise computations," in *RTCSA*, 2020.
[26] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016.
[27] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: Energy-efficient neuromorphic systems using approximate computing," in *ISLPED*, 2014.
[28] M. Cho and K. G. Shin, "Dynamix: Resource optimization for dnn-based real-time applications on a multi-tasking system," *arXiv*, 2023.
[29] M. V. Beigi and G. Memik, "Thermal-aware optimizations of reram-based neuromorphic computing systems," in *DAC*, 2018.
[30] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016.
[31] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *HPCA*, 2017.
[32] S. Lee and S. Nirjon, "Subflow: A dynamic induced-subgraph strategy toward real-time dnn inference and training," in *RTAS*, 2020.
[33] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference," in *RTSS*, 2019.
[34] K. Qiu, W. Chen, Y. Xu, L. Xia, Y. Wang, and Z. Shao, "A peripheral circuit reuse structure integrated with a retimed data flow for low power rram crossbar-based cnn," in *DATE*, 2018.
[35] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *ASPLOS*, 2020.
[36] B. Kim, S. Lee, A. R. Trivedi, and W. J. Song, "Energy-efficient acceleration of deep neural networks on realtime-constrained embedded edge devices," *IEEE Access*, 2020.