Hierarchical Memory System With STT-MRAM and SRAM to Support Transfer and Real-Time Reinforcement Learning in Autonomous Drones

Insik Yoon, Malik Aqeel Anwar, Rajiv V. Joshi, *Fellow, IEEE*, Titash Rakshit, and Arijit Raychowdhury, *Senior Member, IEEE*

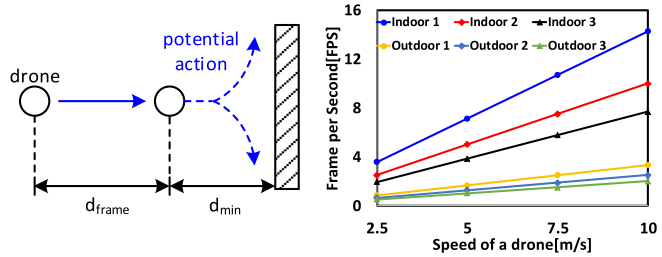
***Abstract*— This article presents a transfer learning (TL) followed by reinforcement learning (RL) algorithm mapped onto a hierarchical embedded memory system to meet the stringent power budgets of autonomous drones. The power reduction is achieved by 1. TL on meta-environments followed by online RL only on the last few layers of a deep convolutional neural network (CNN) instead of end-to-end (E2E) RL and 2. Mapping of the algorithm onto a memory hierarchy where the pre-trained weights of all the conv layers and the first few fully connected (FC) layers are stored in dense, low standby leakage Spin Transfer Torque (STT) RAM eNVM arrays and the weights of the last few FC layers are stored in the on-die SRAM. This memory hierarchy enables real-time RL as the drone explores unknown territories and the system only reads the weights from eNVM (that are slow and power hungry to write otherwise) for inference and uses the on-die SRAM for low latency training through both write and read of the weights of the last few layers. The proposed system is extensively simulated on a virtual environment and dissipates 83.5% lower energy per image frame as well as 79.4% lower latency as compared to E2E RL without any loss of accuracy. The speed of the drone is improved by a factor of 3× due to higher frame rates as well.**

***Index Terms*— Transfer learning, reinforcement learning, autonomous drone, object avoidance, deep learning, convolutional neural network, STT-MRAM.**

I. Introduction

OVER the past decade, applications such as reconnaissance, surveying, rescuing and mapping with Unmanned Aerial Vehicles (UAVs) or drones have achieved substantial success. For all these applications of UAVs, navigating autonomously in varied environments with camera based inputs is considered a key enabling feature. Recently, reinforcement learning (RL) on robotic tasks such as real-time drone navigation and collision avoidance has been extensively explored [1], [2]. However, online, real-time RL continues to be computationally challenging despite its recent success and its bio-mimetic approach. In typical RL systems, a deep convolutional neural network (CNN) is used to achieve a functional mapping images (system states) to the best possible action. In the case of RL for real-time collision avoidance, a major latency bottleneck arises from the need to train a CNN with the current image frame, which must be completed before the next image frame is captured [2], [3].

This is illustrated in Fig. 1 where we show the relationship between the speed of a drone and the required frame per second (fps) of the image acquisition system. As shown in Fig. 1(a), For a given velocity of the drone, we can calculate the minimum fps requirement of the camera for collision avoidance based on the corresponding distance traveled between two frames (d_{frame}), and the minimum distance between the drone and its obstacles (a measure of clutter in the environment). From Fig. 1(b), we observe that the fps requirement increases as the speed of a drone increases. Since the minimum distance between a drone and its obstacles is lower in typical indoor environments compared to outdoor



(a) (b)

Environment	d_{min}[m]
Indoor 1	0.7
Indoor 2	1
Indoor 3	1.3
Outdoor 1	3
Outdoor 2	4
Outdoor 3	5

	Image frame per second[FPS]					
v_{drone}[m/s]	Indoor 1	Indoor 2	Indoor 3	Outdoor 1	Outdoor 2	Outdoor 3
2.5	3.571	2.5	1.923	0.833	0.625	0.5
5	7.142	5	3.846	1.666	1.25	1
7.5	10.71	7.5	5.769	2.5	1.875	1.5
10	14.28	10	7.692	3.333	2.5	2

(c)

Fig. 1. (a) Definition of minimum distance required for obstacle avoidance (d_{min}). d_{frame} = distance that drone moves between frames. (b) Frame per second vs. speed of a drone for sample indoor and outdoor environments (c) d_{min} setting for different environment and minimum FPS needed for obstacle avoidance for different environments [4].

Manuscript received April 15, 2019; revised June 26, 2019; accepted July 27, 2019. Date of publication July 31, 2019; date of current version September 17, 2019. This work was supported by the Semiconductor Research Corporation under Grant JUMP CBRIC task ID 2777.006 and Grant JUMP ASCENT task ID 2776.004. This article was recommended by Guest Editor K. Kailas. *(Corresponding author: Insik Yoon.)*

I. Yoon, M. A. Anwar, and A. Raychowdhury are with the Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: iyoon@gatech.edu; aqeel.anwar@gatech.edu; arijit.raychowdhury@ece.gatech.edu).

R. V. Joshi is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: rvjoshi@us.ibm.com).

T. Rakshit is with the Advanced Logic Lab, Samsung Semiconductor, Austin, TX 78754 USA (e-mail: titash.r@samsung.com).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2019.2932285

environments (i.e., the indoor environment is more cluttered than outdoor environments), drones in the indoor environment require higher fps compared to outdoor environments. As the fps increases, the time available to perform real-time RL decreases necessitating high-performance of the computing system. For small power-constrained drones, it requires significant hardware resources to execute the training process in RL within the latency and power targets. Further, embedded non-volatile memory (eNVM) [5]–[7] has emerged as a potential candidate for DRAM replacement for its high density and low standby power. This is particularly useful to store model weights of CNNs that can achieve RL in embedded systems, such as small drones. However, all eNVM technologies, including Spin Transfer Torque Magnetic RAM (STT-MRAM) exhibit high write latency and energy and does not meet the write energy and latency targets for real-time RL.

To address this fundamental challenge, we propose an algorithm-hardware co-design where we show:

1. Context-aware transfer-learning (TL) augmented with RL. Before deployment, an agent (drone) is trained in complex meta-training-environments (indoor and outdoor) in a virtual simulation platform. During training, the agent captures an image of an environment (called state) and a CNN provides the optimal action based on the current state to maximize some notion of long-term reward. Training the policy network is accomplished via RL. Once the system reaches the target performance, the trained weights of CNN in complex meta-environments are ready to be transferred to a drone (Transfer Learning) at the time of deployment.

2. At the time of deployment, the correct meta-model (indoor or outdoor model) obtained from TL is downloaded to the drone. In our studies, we consider a prototypical embedded platform consisting of a large, stacked-eNVM array and a smaller (30 MB) on-die SRAM. As a part of this study, we consider spin-transfer-torque (STT-RAM) as the NVM of choice. A part of the model (last few layers of the neural network) are stored in the on-die SRAM and rest of the model is stored in STT-MRAM stack.

3. After deployment, the drone performs real-time RL; the drone first reads the weights stored in NVM to perform inference to determine the best action (forward propagation of the CNN) based on the current state (acquired image). Once the drone receives the next state after the execution of inferred action, RL evaluates the error and train the weights in CNN. But instead of learning all the weights in every layer of the CNN, the system only trains the last few layers of CNN whose weights are stored in the on-die SRAM. This results in only read accesses from the e-NVM array during flight (inference/ forward propagation of data) and all the necessary write operations are executed on the on-die SRAM. In the process of inference (forward propagation of data), the system only reads the weights of the model from the eNVM to the SRAM. The weights of the last few layers stay in the SRAM and the updated weights of the last few layers are written to SRAM at the end of training. We also show a typical case where a small portion of the weights stored in the STT-MRAM array is updated in real-time. Since the convolution layers of the network stores the coarser features of the environment (obtained from TL), the proposed algorithm works successfully as the drone needs to learn only the environment specific finer features (online RL) in real-time. We show that the using TL followed by environment-specific RL over the last few layers achieves comparable accuracy as E2E RL. While E2E RL on an environment is not feasible with NVM based embedded platforms (in terms of latency and energy requirements), our proposed solution archives real-time operation with 79.4% (83.45%) decrease in latency (energy) in PE array compared to a baseline E2E RL system. Due to the stringent power constraints of a drone, the system employs STT-MRAM instead of DRAM because using STT-MRAM can save the amount of energy used for refresh operation from DRAM since refresh operation is not required in STT-MRAM. With 83MB of weights stored in STT-MRAM, dissipated energy over 1000 iterations of STT-MRAM presents 58% decrease compared to the energy dissipated from DRAM in the case of on-line training of last 4 layers.

II. Reinforcement Learning for Drone Navigation

A. Basics of Reinforcement Learning

Before going into the details of the platform architecture, let us briefly review RL in the context of autonomous flight in small form-factor drones. Reinforcement learning (RL), inspired by behavioral psychology, learns by interacting with the environment in discrete time steps [1], [8]–[10]. As opposed to supervised learning, RL doesn't have direct access to the data labels. The labels for RL can be thought of as dynamic and are generated and updated online until convergence is achieved. The agent is placed in the training environment and is allowed to take actions to explore the environment. With every action taken, the agent is presented with a reward based on a user defined goal. The reward quantifies the underlying goal; if the agent took an action that was in accordance with the goal, the reward would be higher and vice versa. The objective of RL is to learn a control policy that predicts actions maximizing these long-term rewards. For the case of autonomous flight, the RL problem will be formulated as follows. The goal is to avoid crashing into the obstacles, hence the notion of distance between the drone and the nearest obstacle can be used as a reward. A set of feasible actions is defined for the action space (in our case moving forward, moving left and moving right). The agent is only allowed to select among these set of actions. Resized RGB images from the drone's camera are used as states. Once the goal, state and action space are defined, the agent is placed in the training environment. At time step t, the drone observes the current state s_t, takes an action at from the action space and moves to a new position and observes a new state s_{t+1}. These current and new state pair along with the actions taken are used to generate a reward r_t (s_t,a_t, s_{t+1}). For each step, these four quantities together define an RL data-tuple (s_t,a_t,s_{t+1},r_t). The objective of RL is to predict set of subsequent actions, leading to the maximization of the long-term discounted return

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$$

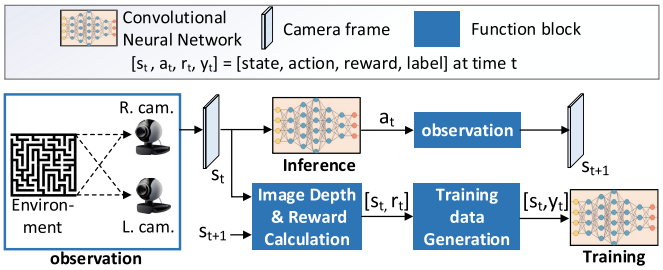
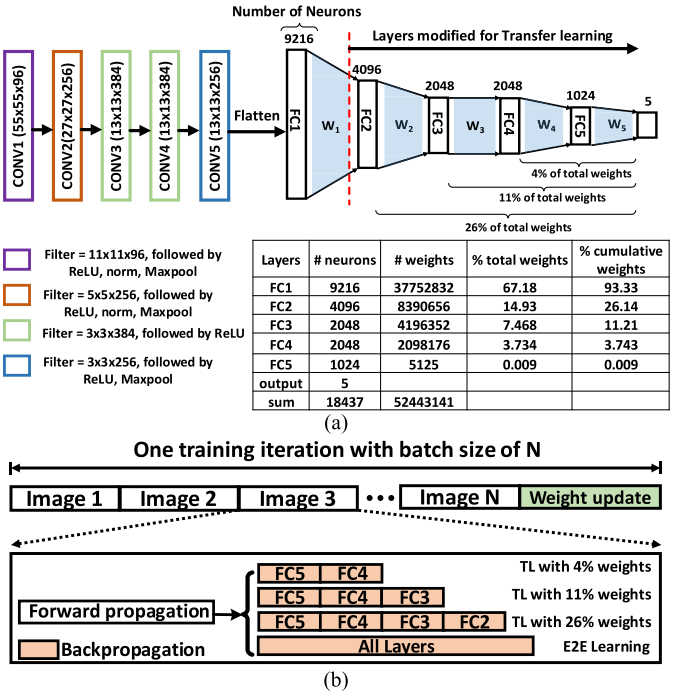


Fig. 2. Reinforcement Learning(RL) network architecture for camera based navigation in drones [4].



Layers	# neurons	# weights	% total weights	% cumulative weights
FC1	9216	37752832	67.18	93.33
FC2	4096	8390656	14.93	26.14
FC3	2048	4196352	7.468	11.21
FC4	2048	2098176	3.734	3.743
FC5	1024	5125	0.009	0.009
output	5			
sum	18437	52443141		

Fig. 3. Reinforcement Learning(RL) network architecture for camera based navigation in drones. (a) Modified AlexNET [14] for the proposed system (b) 3 configurations where 4,11 and 26% weights are learnt in real-time. This is in contrast to E2E RL, where the entire network is learnt in real-time. [4].

where, γ is the discount factor). This is done by converting the data-tuples into sets of training pairs. The effectiveness of taking an action a_t from a given state s_t is quantified by its corresponding Q-value $Q(s_t,a_t)$. The greater the Q-value, the more favorable the action is. These Q values are updated online using the Bellman Equation

$$Q(s_t, a_t) = r + \gamma \, max_a Q(s_{t+1}, a) \tag{1}$$

The training pairs $(s_t,Q(s_t,a_t))$ are then used as the input-output pairs for training the network. At any given state, the network predicts the action with the maximum Q value $a' = max_a Q(s_t, a)$. RL for obstacle avoidance and path-planning has been successfully applied in prototypical robotic vehicles [11], [12] and in Parrot AR drones [13] and interested readers are pointed to [12] for a detailed overview.

B. RL in Camera Based Navigation in Drones

We focus on the implementation of a camera based drone system that performs end-to-end navigation via collision avoidance (long term goal) as shown in Fig.2. The navigation problem is mapped to the RL problem as follows. The state at time instant t, $s_t \in S$ is the image frame of the environment from the camera. At any given state, the drone takes any action $a_t \in A$ where A is the action space. In this proposed system, the action space is limited to have five values $A = 0, 1, 2, 3, 4$. 0 in action space A indicates that the drone moves forward, 1 and 3 mean that the drone turns left with turn angles 25 degree and 55 degree respectively. Similarly, 2 and 4 means turning right with turn angles 25 degree and 55 degree. These five actions are sufficient for the drone to navigate in its surrounding. When the image frame is captured from the stereo camera, a disparity map is used to generate an approximate depth map of the image frame [2]. From the generated depth map, a reward is generated in a manner described in [3]. In the process of reward generation, the depth map is segmented into smaller window at the center and the average depth of this center window correlates to the value of reward. Therefore, the reward becomes smaller when the drone is closer to obstacles because the average depth in the center window is less. The Q values for the states are estimated using a deep convolutional network (CNN). The image frame obtained from the camera is the state at time t, $s_t \in R^{nxn} where n = 224$ and becomes an input to the CNN.

In order to have the network architecture optimized for autonomous navigation, we modified the AlexNet model [14] and used it as the CNN. It consists of 5 convolutional layers and 5 fully connected layers. The detailed network architecture and parameters are shown in Fig.3. During the online RL when the drone is flying, the CNN learns the weights of the model and keeps on improving the functional mapping from the state to the action.

C. Challenges of End-to-End(E2E) RL in Embedded Systems

In a true biologically-inspired system, an autonomous drone should learn to navigate via E2E RL, where the reinforcement learning algorithm trains the weights in every layers of the CNN [3]. From starting with randomly initialized weights of the model, the drone should learn the model that efficiently maps state to action from iterative interactions with the environment. Although feasible [3], this faces two fundamental challenges:

1. During exploration, the drone will take random actions and they are often incorrect actions, especially at the beginning of the flight. This can lead to collisions with obstacles. The sequence of incorrect action that lead to collisions can cause damage to the drone or the environment.

2. Further, E2E RL is computationally extremely challenging. Since E2E RL requires training of all weight parameters in every layers of CNN, it is almost impossible to achieve autonomy via RL in small form factor drones, without additional off-board infrastructure [3]. Improvements in both design [14], [15] as well as technology [16]–[21] continue to make CNNs a reality on resource constrained edge-devices. In particular, eNVM is a promising replacement for DRAM to achieve high energy-efficiency. Among competing eNVM

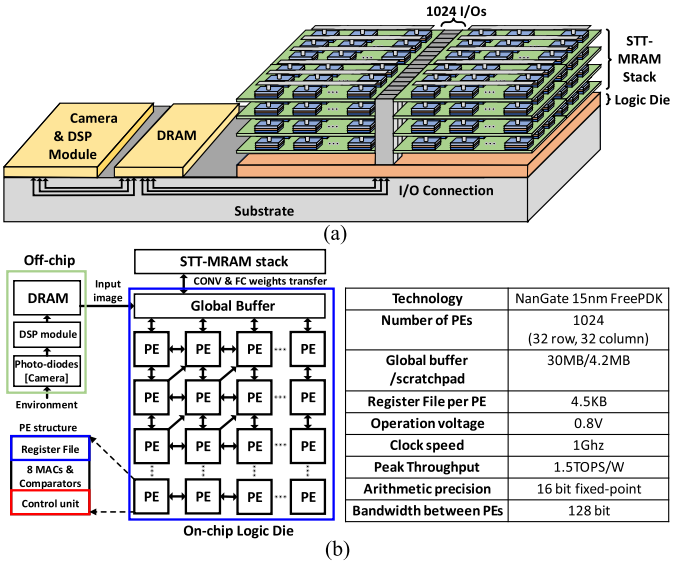
technologies, such as RRAM [17], [21], PCRAM [19], FerroFETs [20], STT-MRAM [18], [22] is considered more mature and exhibits high density, endurance and nano-second read speeds. However, the write latency and energy of STT-MRAM is expensive and is a major bottleneck in real-time RL and continuous weight updates.

III. Proposed Approach Using Transfer Learning(TL) With Real-Time RL

In transfer learning, pre-existent knowledge of the source tasks from one or more domain is used to learn target task in another domain. Transfer learning approach to solve various problems in deep learning has been there for over a decade. It has been used in the past for the purpose of mitigating convergence issue, faster convergence, improving target performance, reducing the time of convergence and addressing the issue insufficient data [23]–[25], where the weights of the deep network learnt for one problem is used as initial weights for some another similar problem. The network is then fine-tuned, end-to-end on the new data set converging faster. It is a well understood fact that [26], for a complex enough task, deep network's performance increases by increasing the number of hidden layers (given the amount of training data scales too). So, for an acceptable performance, the network should be deep enough, which comes with additional computational cost. This increased computational cost requires heavy computational resources (like GPUs) and cannot be executed on a resource constrained system/edge node (say a drone). To the best of our knowledge all the TL papers in the past discuss TL as tool/approach to address the above-mentioned issues without worrying much about the computational cost required to train a deep neural network. In this paper we show we can use Transfer learning, to segment a deep network into trainable and non-trainable part reducing the training computations, for underlying task without compromising too much on its performance. We use transfer learning with real-time RL as an algorithmic solution that maps to a hierarchical memory system consisting of stacked STT-MRAM and on-die SRAM. This alleviates the challenges of E2E RL and enables a practical hardware solution to realize autonomous flight with environment specific RL. In our proposed system the agent learns on an embedded platform in the following steps:

1. The CNN is first trained in complex meta-environments in simulation. Once the training is finished, the pre-trained CNN model is downloaded to the system memory as a meta model. We explore two types of meta-environments: outdoor and indoor. Other types of environments can be added depending on the types of real environments that drone is expected to be deployed in.

2. The downloaded meta model is located in STT-RAM and the weights of the last few layers of the CNN are transferred to an on-die SRAM. During real-time learning, the system reads the weights of each layer of STT-MRAM to SRAM for inference and once inference is finished, we train the weights of the last few fully connected (FC) layers of the model and write the updated weights back to the SRAM. By performing inference with the weights from TL and training the last few



Technology	NanGate 15nm FreePDK
Number of PEs	1024 (32 row, 32 column)
Global buffer /scratchpad	30MB/4.2MB
Register File per PE	4.5KB
Operation voltage	0.8V
Clock speed	1Ghz
Peak Throughput	1.5TOPS/W
Arithmetic precision	16 bit fixed-point
Bandwidth between PEs	128 bit

Fig. 4. (a) 3D view of the hardware platform (b) System architecture and parameters as extracted post-synthesis in 15nm Nangate PDK. [4].

fully connected layers of the network via RL, we can reduce the latency and energy of the system significantly. This extends the drone's battery life and enables the system to support a higher speed as illustrated in Fig. 1. Fig. 3(a) presents three different architectures for training. Based on the on-die SRAM capacity, we can store 26% (FC2+FC3+FC4+FC5), 11% (FC3+FC4+FC5) and 4% (FC4+FC5) of the total weights of the network in the SRAM. The procedure of on-line training is described in Fig. 3(b). In order to complete one training iteration with batch size of N images, the system performs N number of computation, which is defined as taking one image at a time and complete forward and backward propagation. In the following sections, we compare the system performance of TL followed by RL, which train the last 2/3/4 layers of the network, and E2E learning (baseline), the algorithm that trains all parameters in the network.

IV. Proposed System Architecture

The system architecture includes a logic die that contains of a systolic array of processing elements [14] and a global buffer (on-die SRAM) and a STT-MRAM stack on top of the logic die (Fig. 4). We assume that the architecture of the sub-array organization and local/global IO of STT-MRAM stack is same as the DRAM-based High Bandwidth Memory (HBM) architecture from JEDEC [27]. The DRAM subarrays of DRAM-based HBM are replaced with STT-MRAM. By using STT-MRAM in the DRAM-based HBM architecture, we provide a realistic and emerging platform for an embedded system with high-bandwidth IO, based on HBM JEDEC specification [27]. Recent progress in manufacturing, design and packaging [28]–[31] promises decreasing the feature size of the STT-MRAM bitcell through improvements in nano-patterning of MTJ, design of the selector transistor and optimized peripheral circuits. In a recent demonstration [15], a monolithic design of a 4Gb array featuring a $9F^2$ cell was

shown. This shows that even a single layer of 3D-stacked STT-MRAM can feature more than 100MB of capacity in a single stacked layer.

One of the concerns in high-density compute in 3D stacks is the elevated temperature of the stacks. Since the STT-MRAM layer is located directly above the logic die, the temperature from the logic die can affect the retention of 3D-stacked STT-MRAM and may cause bit flips in the array. In order to prevent erronous bit flips caused by elevated temperature, the free layer (bit storage layer) of the STT-MRAM must be thick enough to guarantee non-volatility even at elevated temperatures as discussed in [32] . Correspondingly, there will be an increase in write critical current and power of STT-MRAM, which is an issue of allocating enough design margin. [33] and [34] present 10 year retention at 90 and 85 degree Celsius. A complete thermal design and modeling for 3D chip-stack is outside the scope of this paper. There has been significant work on modeling 3D chip-stacks and research results [35]–[38] in this area will guide both placement as well as routing of PEs and other functional units.

A system with camera, image processing DSP module and DRAM buffer memory is integrated on a substrate (which can be a silicon interposer or a package substrate) as shown in Fig. 4(a). The connections between each module and the logic die are assumed to be DDR6 links.

A. Off-Chip to On-Chip Data Movement

The camera with a DSP module and buffer-DRAM are located off-chip on a shared substrate. Once an image is captured by the camera, the DSP module resizes the image to 224 by 224 as described above and stores the output to a DRAM memory buffer. The image is serially read from DRAM buffer to the logic die as input to the CNN and stored in the on-chip global buffer. During the inference process, the image from global buffer is distributed to the register files in the PE array.

B. On-Chip System Architecture With Stacked STT-MRAM

The logic die that contains of the spatial PE array and a global buffer located on a common substrate [27] and 3D-STT-MRAM [6], [7] is stacked on top of the logic die in the same way as DRAM-based HBM is currently stacked. STT-MRAM stack is used as a weight storage and it contains all weights from each layers of the network. The systolic array of PE has 1024 PEs in total (32 rows, 32 columns) and the bit width of the connections between PEs is 128 bit. One a PE is connected with 5 nearby PEs (top, bottom, left, right and upper right) [14], [15]. The bit width of the connections between the global buffer and the 32 PEs at the first row of the PE array is 4096 and the global buffer can broadcast the same data to each PEs in the first row. STT-MRAM stack has 1024 I/O connections (each I/O has 2Gbit/s of bandwidth) with the global buffer [27]. Each PE has a register file, 8 MACs for convolution and vector-matrix multiplication and 8 comparators for rectified linear and maxpool operations. Fig. 4(b) shows a complete list of system parameters. The whole system is designed, synthesized and in the 15nm nangate technology [39]. All results discussed here are post-synthesis.

TABLE I
COMPARISON BETWEEN STT-MRAM [16], [18], [41], [42] AND COMPETING TECHNOLOGIES (EFLASH [43]–[45], RRAM [17], [21], PCRAM [19], [46]

	SRAM	Eflash	STT-MRAM	RRAM	PCRAM
Cell size	80~100F^2	~6F^2	>6F^2	>4F^2	>4F^2
Non-volatility	No	Yes	Yes	Yes	Yes
Program voltage	< 1V	<~10V	< 1.5V	< 3V	< 1V
Write speed	~1ns	660 µs	~30ns	~ 1 µs	~80ns
Read speed	~1ns	45 µs	~10ns	~1 µs	~10ns
Endurance	10^{16}	$10^4 \sim 10^6$	10^{15}	10^{10}	10^{12}
Retention	N/A	10 yrs	10 yrs	10 yrs	10 yrs

C. Motivation for STT-MRAM and STT-MRAM Basics

It is well understood that next-generation memory-intensive ultra low power learning-based systems require a memory technology which shows 1. high-density, 2. low-standby power (hence eNVM) 3. acceptable R/W speeds and importantly 4. compatibility with a logic process both in terms of process thermal budget and voltage domains. This is required to ensure that the design, along with an eNVM, can take advantage of the numerous scaled high performance, low power digital logic blocks that are essential for any area and power constrained design like the one we have described in this paper. Compared to other NVMs such as Phase-change memory or resistive RAM, STT-MRAM exhibits better read/write latency [15], [16] and is more mature than Ferroelectric FET based RAMs. Recent publications from leading foundries [7], [18], [22] have demonstrated MBs of STT-MRAM arrays with necessary peripheral circuits. Compared to STT-MRAMs, RRAMs show larger device-to-device and cycle-to-cycle variations making it hard to commercialize [40].

Although our study investigates STT-MRAM based stacks, all eNVM suffer from high write latency and energy; and hence the algorithm-hardware co-design that we propose is applicable to similar other platforms. The STT-MRAM model parameters are summarized in Table I.

The STT-MRAM bitcell consists of one access transistor and one Magnetic Tunnel Junction (MTJ) where a single bit of information is stored. An MTJ is formed with two ferromagnetic CoFeB based layers and one insulating layer (MgO) in between [18]. One ferromagnetic layer is called a fixed layer because its magnetic moment is fixed to one direction. The other ferromagnetic layer is called a free layer since the direction of magnetic moment can be changed based on the direction of current flowing across the MTJ.

Fig. 5 describes how the direction of magnetic moment in the free layer changes based on the current across the MTJ. Fig. 5 shows how the direction of magnetic moment in the free layer changes from (a) anti-parallel to parallel and (b) parallel to anti-parallel direction compared to the direction of magnetic moment in fixed layer. Since the fixed layer acts as a spin polarizer, the spin polarized electrons that pass the fixed layer exerts the torque on the magnetic moment in the free layer and causes a flip in the direction of the magnetic moment in fixed layer as shown in Fig. 5(a). When the current flows

Fig. 5. The direction of magnetic moment in free layer changes from (a)anti-parallel to parallel (b) parallel to anti-parallel to the direction of magnetic moment of fixed layer. The arrow in the free/fixed layer indicates the direction of magnetic moment.

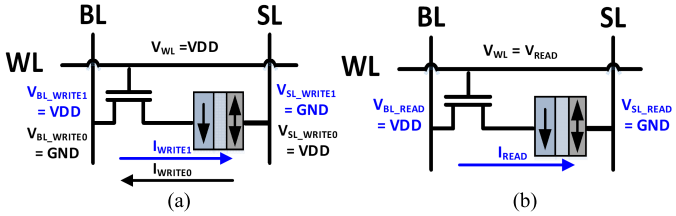


Fig. 6. The STT-MRAM cell schematic of (a) write (b) read operation.

TABLE II
STT-MRAM ARRAY PARAMETERS COMPILED FROM [18], [22], [47]

Technology	22nm FFL FinFET
TMR	180%
RA	9 $\Omega\mu m^2$
Density	8Mb
Cell architecture	1T 1MTJ
unit cell size	$9F^2$
Power supply(core)	1.0V
MTJ size	60~80nm
MTJ type	Perpendicular MTJ

from the fixed layer to the free layer as shown in Fig. 5(b), the electrons with opposite spin are reflected back from the fixed layer and exerts a torque that changes the direction of the magnetic moment of the free layer to an anti-parallel direction with respect to the magnetic moment in the fixed layer. The alignment of the magnetic moment in the fixed and free layers determine the resistance across the MTJ. When the magnetic moments in the two layers are anti-parallel to each other, the resistance across MTJ is high.

A low resistance is achieved when both the magnetic moments are parallel to each other. The high/low resistance is mapped to 1/0. The bias conditions applied for the write and read operations are shown in Fig. 6. As shown in Fig. 6(a), the write operation is bi-directional. In case of writing a 1, the bit-line and the source line are set to VDD and GND and the write current flows from the fixed layer to the free layer of the MTJ. The biasing condition for writing a 0 is the opposite and is shown in Fig. 6(a). In case of read operations, the word-line is asserted to VREAD and the bitline and the source line are set to VDD and GND. This causes a weak current to flow across the MTJ and the resistance state is sensed using either a constant current scheme or a BL discharge scheme [48]. Table II shows STT-MRAM array parameters from the silicon implementation of STT-MRAM.

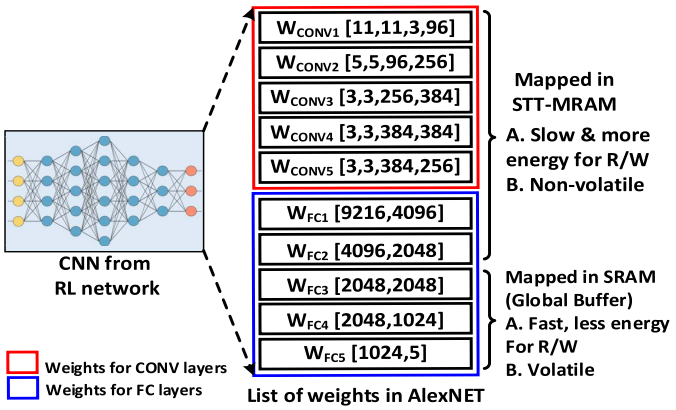


Fig. 7. Mapping the weights of the proposed CNN (modified AlexNET) to stacked-STT-MRAM and on-die SRAM in the system [4].

D. Mapping the CNN Model to the Memory System

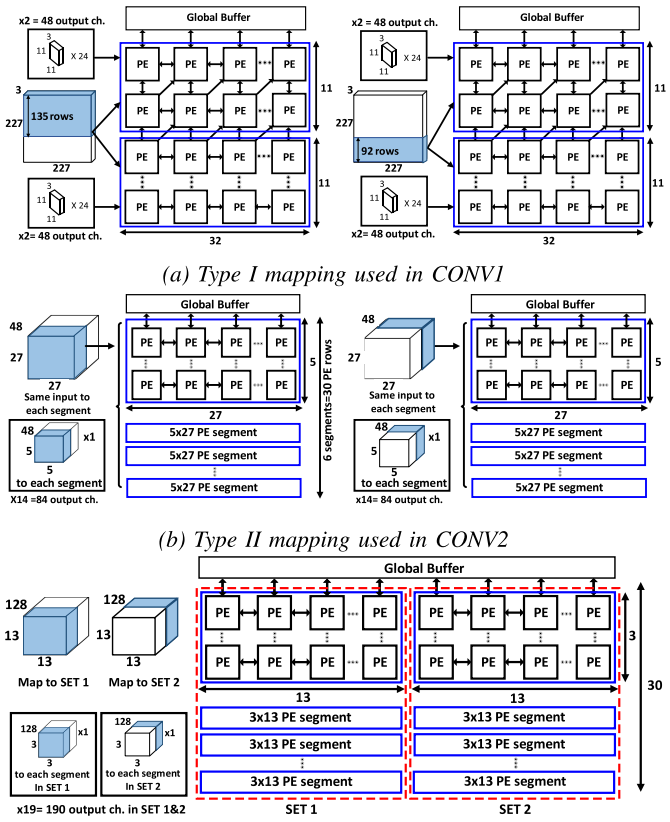
Fig. 7 presents how the model weight of the CNN is mapped to the memory system comprising of the stacked-STT-MRAM and on-die SRAM. The size of on-chip SRAM-based global buffer must be large enough to store the weights of the last 2/3/4 fully connected layers of the network since the system performs real-time update of these weight parameters inside the global buffer. Since each parameter is 16 bit fixed point, the size of the SRAM should be 29.38MB if we store all weights from the last 4 fully connected layers. In the proposed design, we store the weights from the last three layers to the global buffer and the sum of all weights of the last three layer is 12.6MB. The rest of the weights from all convolutional (CONV) layers and the first and the second fully connected layers (FC1, FC2) add up to 100MB and they reside in the STT-MRAM array. In addition to this, the weight and bias gradients of the last 3 layers of the network are stored in the global buffer for the weight update in RL. Once we have the sum of gradients of weights and bias after processing a batch size of N, we need to update the weights as shown in a manner shown in Fig. 3(b) and this requires an additional 12.6MB of global buffer. In summary, the global buffer uses 25.2 MB of space to store weights of the last three layers for forward propagation and the sum of the weights and bias gradients from the last three layers used during backpropagation. Lastly, scratchpad for loading/storing intermediate results, input and weight parameters to the PE array takes 4.2MB of space in the global buffer. In summary, we need on-chip SRAM size to be 29.4MB, which is at-par with the on-die SRAM capacity of practical embedded systems.

V. FORWARD PROPAGATION THROUGH THE CNN

A. Forward Propagation in Convolution (CONV) Layers

A row stationary dataflow architecture is used in the systolic array for convolution during forward propagation [49]. The basic steps are:

1. Input image to the convolution layer is loaded from the global buffer to the local register file (RF) in each PE. Once the input image is stored in the RF of each PE, the row of



(a) Type I mapping used in CONV1

(b) Type II mapping used in CONV2

(c) Type III mapping used in CONV3,CONV4, CONV5

Fig. 8. Strategies for mapping weights and data for processing the convolutional layers [4].

the image is transferred to the nearby PEs by using diagonal connection to maximize data reuse within the PE array.

2. Each row of filter weights is broadcasted from the global buffer to the RF in each PE in the same row of the PE array.

3. MAC units inside each PE computes row-wise convolution of image row and filter row and the result of the convolution (pSUM) is stored in the RF.

4. pSUM from each PEs in the same column are accumulated vertically to the PE in the first row and the accumulated values from the first row of the PE array are written back to the global buffer

In order to effectively utilize the hardware resource for computing convolution, we have three ways of partitioning PE arrays into segments based on the height of the filter in CONV layers.

Based on the partitioning, the data mapping of the filter weights and the input are determined. The major factors that determine the partitioning the PE array are the size of RF inside the PE, the dimension of PE array and the filter size of the CONV layers. Fig. 8 shows all types of partitioning of PE arrays and the corresponding data mapping techniques. Fig. 8a shows how Type I partition is applied to the first convolution (CONV1) layer, whose filter dimension is (11,11,3,96) and stride is 4. In CONV1, each row of filter and image data with all input channels can fit into the RF of each PE in the same row. The PE array is partitioned into two segments whose dimensions are 32×11. Since the height of the filter is the same as the height of the segment, each row of the filter is mapped to each row of the PEs in the segment. The same image data is loaded to two segments of the PEs and filters with 24 different output channels are mapped to each segment. Depending on the RF size, the number of output channels of the filters can vary. The number of columns inside the segments is equal to the number of rows of images that the system can convolve per cycle. For example, TYPE I configuration produces the convolution results of 135 rows of input image in a single cycle. (135 = 32*stride + filter height) because the number of columns in the segment is 32. Fig. 8b presents the TYPE II mapping scheme of data for the second convolution layer (CONV2). In this case, the RF in a PE cannot fit the row of the image and the filter with all input channels because the data size is too large. Therefore, TYPE II divides input channels of filter and images into two parts and loads them into segments of the PE array. Since the filter height of CONV2 is 5, the dimension of each segment is 27×5 and the PE array is partitioned into 6 segments. Instead of using all 32 columns of PE, 27 columns are utilized because each column generates one row of convolution output. The same image data is mapped to all 6 segments and each segment is mapped with the corresponding filters and each segment generates distinct outputs at the end of computation. Fig. 8c presents the TYPE III mapping scheme of data for CONV3. The main difference between TYPE II and TYPE III mapping is the existence of set, which is defined as a cluster of PE segments. Since the filter width and height decreases from CONV2 to CONV3, we can map 2 sets of 10 segments (each segment dimension is 3×10 PE) to PE array for CONV3. In the TYPE III mapping scheme, the segment size of the PE is 3×13 because the filter dimension is (3,3) and the stride is 1. Because the dimension of the segments is lower, we partition the PE array into 2 sets of 10 segments (total 30×26 PE array). Due to the high number of input channels of input and filter to CONV3, we split the input channel of filter and inputs into two parts. Unlike TYPE II, the two parts of inputs and filters are mapped to each set of the PE array, which enables us to map the input and the filter with all the input channels. After completing pSUM in step 4, the convolution results in the first row of set 2 must be transferred to the first row of set 1. For example, the output from PE at 14th column (PE in the 1st column in set 2) must be transferred to the PE in the 1st column in set 1. Then the two results from set 1 and set 2 are added together to complete the convolution. Since the filter height and width (3,3) in CONV4 and 5 are the same as the filter height and width in CONV3, the TYPE III mapping scheme is used for CONV4 and 5 as well.

B. Forward Propagation in Fully Connected(FC) Layers

Vector-matrix multiplication is the core computation in the forward propagation of Fully Connected layers. Fig. 9 describes how the input vector and the weight matrix are mapped to each PE in the array to perform vector-matrix multiplication. Once the values of the weight matrix are loaded to the PE array, the input vector is loaded to the first column of

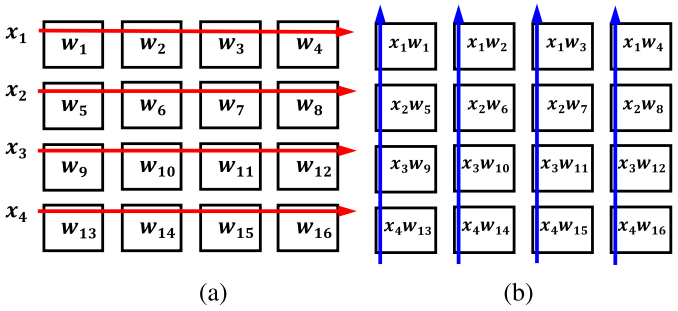


Fig. 9. (a) Row-wise vector propagation in PE array for calculating pSUM (b) Vertical pSUM accumulation for vector-matrix multiplication in forward propagation of FC layers [4].

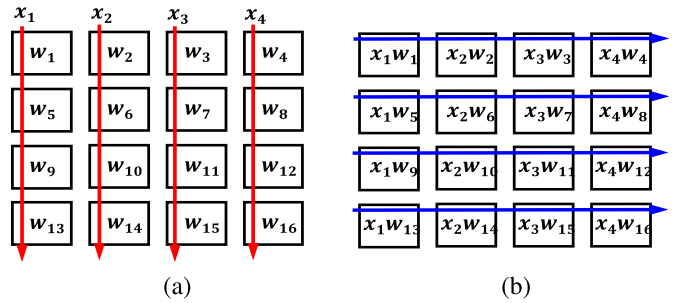


Fig. 10. (a) Column-wise vector propagation in PE array for calculating pSUM (b) Row-wise pSUM accumulation for vector-transposed matrix multiplication in backpropagation of FC layers [4].

the PE array. Then the values in the input vector are propagated row-wise in the PE array and we perform multiplication in each PE. The outcomes of computation (pSUMs) in each PE in the same column are propagated and accumulated vertically. The accumulated results in the first row of PE array are transferred to the global buffer.

VI. Backpropagation and Gradient Descent

For TL followed by online RL, we train last 2/3/4 FC layers of the network. Backpropagation consists of two major computational steps: finding gradients of weights and their biases. Since we use our system to serially process one image at a time for training, the system must store the sum of weight and bias gradient of each image in the global buffer.

A. Backpropagation architecture of Fully-Connected Layer

The gradient of the weight is the result of multiplication of every vector element in a layer of neurons and every vector element in the gradient of the loss function computed with respect to the neurons in previous layer. Since there is no pSUM accumulation involved in calculating weight gradients, the results of multiplication of each PE are directly transferred to global buffer. The gradient of the bias in an FC layer is calculated by multiplying the vector of the gradient of Loss with respect to neurons in previous layer and the transposed weight matrix. The structure of the systolic array enables vector-transposed matrix multiplication without transposing the matrix itself, in a manner describe in [49] Fig. 10 describes the structure of vector-transposed matrix multiplication in the

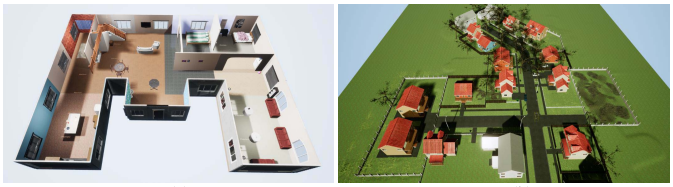


Fig. 11. Screenshots of the complex meta environments developed using UE 4.

PE array. The vector elements are propagated downwards in each column of the array and the pSUM from each PE are accumulated row-wise. The computation is complete when PEs in the last column transfer their results to the global buffer.

B. Backpropagation Architecture of CONV

The backpropagation of CONV layers only happen when evaluating the E2E RL in the system, which is our baseline design. For comparison to the baseline, we benchmark the backpropagation architecture for the entire network. For CONV layers, we use GEMM [50], where the system first reads the data from the STT-MRAM array to the logic die, and expands the inputs to each CONV layers in a 2D matrix. Once the expansion is complete, the backpropagation of CONV becomes same as the backpropagation of FC layers. After the weights of the CONV layers are updated, we write the weights back to the STT-MRAM array. We account for the additional on-chip SRAM requirement for storing the results of the intermediate compute steps.

VII. Simulation Setup

A. Hardware Architecture Simulation

We used NanGate 15nm FreePDK cell library to evaluate the hardware system performance [39]. We perform synthesis and place-and-route of the entire system and the results cited here (along with Fig. 4) are obtained post-synthesis.

B. Simulation Setup

The algorithm is tested on a simulated environment with the dynamics of realistic drones. Simulations were carried out on two types of simulated environments, Indoor and Outdoor. For each of the two categories, complex meta-environments and separate test environments were designed to train and test the performance of the proposed methodology respectively. We used the Unreal Engine 4 (UE4), used for video game development to design the simulation environments and emulate the necessary physics. For each of the two environment categories, a complex meta-environment and two test environments were designed for training and testing purposes. Hence a total of 6 (3 indoor, 3 outdoor) 3D were used in the simulation.

The layouts and screenshots of these environments can be seen in Fig.11 and Fig.12 This engine interfaces with Tensorflow to train a drone via TL and RL. TensorFlow is used as the deep learning framework. AirSim [51] was used to interface the custom generated 3D environments with python. The simulation and training was carried out on a

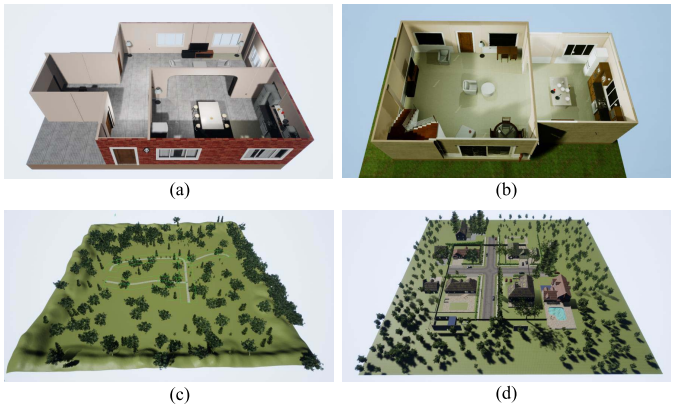


Fig. 12. Screenshots of the test environments (a) Indoor Apartment (b) Indoor House (c) Outdoor Forest (d) Outdoor Town developed using UE 4.

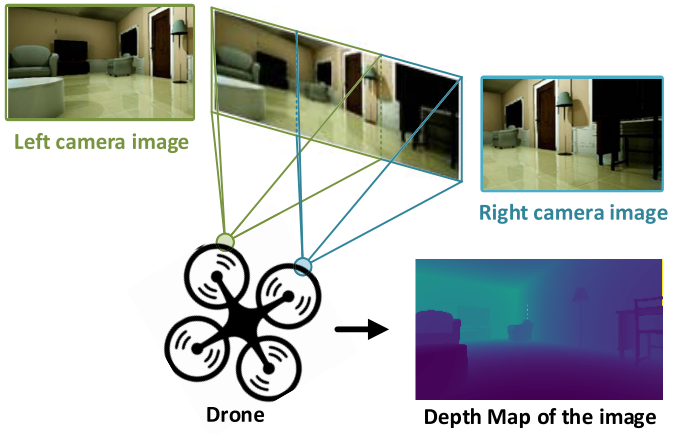


Fig. 13. Stereo vision based depth map generation.

workstation equipped with core i7 processor and NVIDIA GTX1080 GPU. The web-link for the suite of the environments, videos and corresponding data sets can be found here: https://tinyurl.com/y9wgpq4b and the implementation details are beyond the scope of this paper.

C. Training on Meta Environments

The drone is trained in the meta-environment for 60K iterations, initialized with ImageNet [52], [53] weights. For the training, depth maps generated from stereo cameras are used, as shown in Fig.13. The drone is equipped with two cameras (left and right). The scene is captured using these two cameras and the disparity map is generated based on the distance between the corresponding pixels are in the left and right images. The disparity map is passed through a low pass filter to generate a depth map. A typical example is shown in Fig. 13. The training is carried out in two phases. In the first phase the DNN is trained on the complex meta indoor and outdoor environments separately. This DNN is initialized with ImageNet weights (and with random truncated normal weights for the additional layers i.e. FC3, FC4, FC5). In order to help converge the loss, various techniques discussed in [3], such as double deep Q-learning network (DDQN) and clipped temporal difference (TD) error are used. Apart from training the modified AlexNet network for the set of initial weights, the meta-environment is also used to train a small

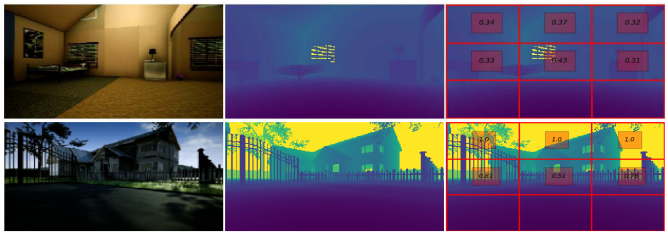


Fig. 14. Feature extraction for SVM classifier On the left, the actual camera frame is shown. The depth map (in the center) is divided into windows and the top 6 windows are used towards feature extraction (right image).

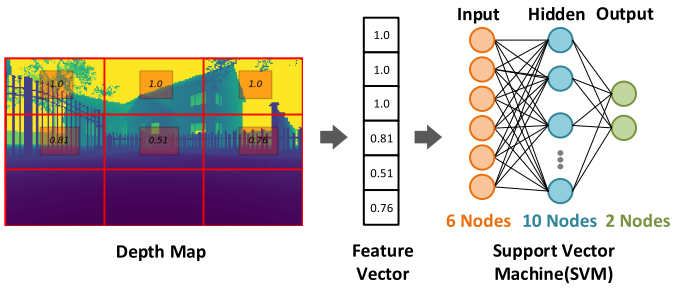


Fig. 15. SVM classifier block diagram.

binary SVM classifier in a supervised manner to differentiate between the indoor and outdoor category of environment. Since outdoor environments typically have objects placed at a larger distance as compared to the indoor environments, the use of the depth map (instead of the raw camera frames) for training the classifier comes as a natural choice. For each of the indoor and outdoor meta-environment, 1000 depth maps are collected. These 2D depth maps are converted into a feature vector of size 6×1 which is used as input to the binary SVM classifier to categorize what category of the environment these depth maps belong to. For each of the 2D depth map the feature vector is generated by slicing the depth map into 9 equal parts. The feature vector is the concatenation of the average of the largest 30% pixel values in the top 6 windows as shown in Fig. 14. The complete block diagram is shown in Fig 14. The classifier is trained on these feature vectors with training accuracy of 98.5% and it is tested on 200 data points from unseen indoor and outdoor environments with an accuracy of 97.02%.

D. Training on Test Environments

Once this training is completed for both the indoor and outdoor meta-environments separately, the transfer learning phase begins. In this phase, for each of the outdoor and indoor category, a DNN is trained for the two test environments separately. The drone is placed in the test environment and uses the trained SVM classifier (Fig. 15) to categorize the environment it is in. The drone collects the depth map by rotating N times with an angle of 360/N degrees. Above mentioned features are extracted from these depth maps and fed to the classifier. Based on the majority label predicted by the binary classifier, the DNN is initialized with the respective trained meta-environment (Indoor or outdoor). Table III lists the hyper parameters used for training. N target is the number of training iteration after which the weights from the target network is copied into primary network in DDQN.

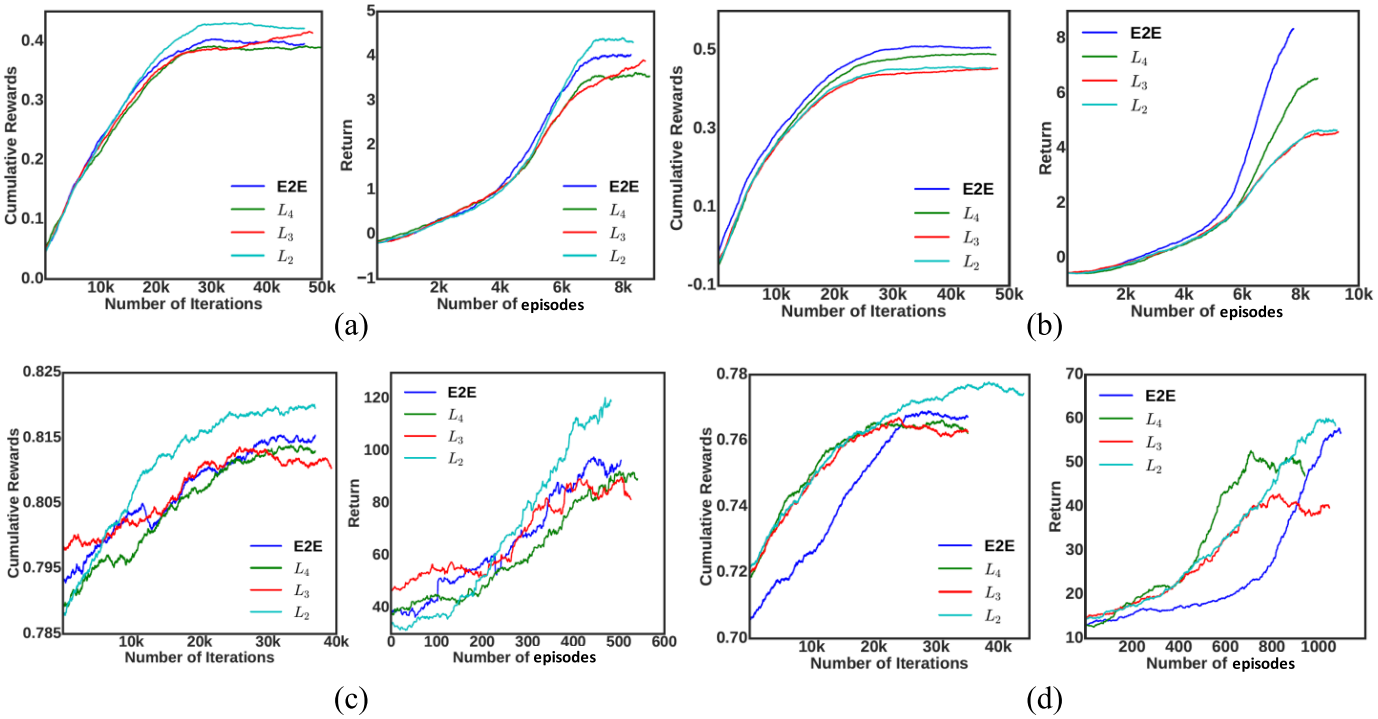


Fig. 16. Cumulative rewards and return results in indoor (a)apartment (b)house and outdoor (c)forest (d)town test environments. The legend L_i indicates TL with last i-layers. All the algorithms show convergence and improving return loss indicating successful learning. [4]

TABLE III
LIST OF HYPER PARAMETERS FOR TRAINING

Learning Rate	N target	Batch size	Iteration
1e-6	500	32	60k

Algorithm 1 Reward Generation Using the Depth Map. The Superscript l,r,c With d Denotes Left, Right, Center Value of Depth Map. the Subscript F,L,R With a Denotes the Forward, Left and Right Action. r and s Are Reward and State. [3]

function $f_r(s_t,a_t,s'_t)$
  $d(s_t) \leftarrow$ *depth map of* s_t
  $d(s'_t) \leftarrow$ *depth map of* s'_t
  $d^l(s_t)$, $d^c(s_t)$, $d^r(s_t)$ = *DepthValues*$(d(s_t))$
  $d^l(s'_t)$, $d^c(s'_t)$, $d^r(s'_t)$ = *DepthValues*$(d(s'_t))$
  if $a_t = a_F$ **then** $r_t = d^c(s'_t)$
  else if $a_t = a_L$ **then** $r_t = d^c(s'_t)+\alpha(d^l(s_t)-d^r(s_t))$
  else $r_t = d^c(s'_t)+\alpha(d^r(s_t)-d^l(s_t))$
  if $d^c(s'_t)<d_{crash}$ **then** $r_t = r_{crash}$
return r_t

Algorithm 1 describes how the depth map is used to generate a reward function for RL with the long-term goal of exploring an area without any collisions. The trained weights from TL are then used as initial weights for RL in the respective test environments For RL, we use 4 topologies, E2E (end-to-end RL) and L_2, L_3, and L_4, where L_i represents TL followed by RL where the last i-layers are trained online.

Fig. 16 reports the results for these test environments in terms of cumulative rewards and return while the safe flight is plotted in Fig 17. Cumulative reward is the moving average of last N rewards received by the agent and is given by $R_i = \frac{1}{N}\sum_{j=i-N}^{i} r_j$ where $i \geq N$ and N is a smoothing constant and was taken to be 15000. The return is the moving average of the sum of rewards across episodes. With each iteration, the agent takes an action and a reward is presented. These rewards are accumulated until the drone crashes and is given by $\frac{1}{N_k}\sum_{j=i-N_k}^{i} r_j$where N_k is the number of actions taken between the k^{th} and $(k-1)^{th}$crash. The return graph from Fig. 16 shows how the learned network performs, on average. Since the goal is not to get to a destination position, but rather to keep on moving around the arena, the return (cumulative reward before crashing) can theoretically become as large as possible. Hence as the system keeps on learning, the return graph will keep on increasing unless the topology itself isn't capable of learning anymore due to limited number of trainable weights. This increase in the return may vary across topologies due to the random nature of the epsilon-greedy exploration [54]. The important takeaway from Fig.16 however, is that the return graph for the topologies with less number of trainable weights (L2, L3) doesn't get saturated at lower value of returns. This signifies superior learning capability of these topologies when initialized with meta-network weights, allowing the proposed technique to have comparable performance as E2E RL. Fig.17 plots the normalized Safe Flight Distance (SFD) across the topologies. The safe flight [3] is the average distance (in meters) traveled by the drone before it crashes and gives a more quantitative measure of how good the drone is in avoiding obstacles. From Fig. 16 we note that the system converges (saturating reward) for all the three scenarios showing the efficacy of the proposed algorithm. The normalized SFD shows acceptable degradation in performance (3% to 8.1%).

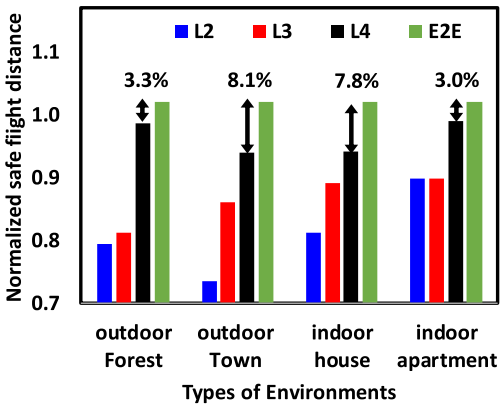


Fig. 17. Normalized safe flight distance (SFD) with respect to different environments. [4]

In outdoor town environments the meta-environment and test environments show large disparities (the type of houses, trees, cars etc. that the drone encounters) and shows the largest degradation. This can be further improved by performing TL on richer meta-environments.

VIII. Hardware Power-Performance Results

The hardware system is evaluated and the post-synthesis results are summarized in Fig. 18 and Fig.19. The latency, energy and number of active PEs for the forward and backward propagation of data for each of the layers is shown in Fig. 18. The major bottleneck of the network layer in terms of processing latency in forward propagation is the first fully connected layer (FC1). Due to the size of its weights (75MB), most of the latency is attributed to data movement; fetching the weights from STT-MRAM to global buffer and distributing it from global buffer to the RF in the PE array. In backpropagation of the last three layers, the system does not access the STT-MRAM because the weights for the last three layers are stored in the global buffer. For backpropagation of the other layers, the weights from STT-MRAM are accessed to find the gradients of input to these layers and to store the gradients of the weights. In Fig. 19, we plot the maximum fps that can be supported in the proposed system vis-a-vis a baseline E2E RL system. We note that for a batch-size of 4, we can support 15fps for L_4, compared to just 3fps for E2E RL. This directly translates to more than 3× increase in the velocity of the drone (Fig. 1). We also achieve a 79.4% (83.45%) decrease in latency (energy) compared to the baseline. While E2E RL is not feasible in terms of energy and latency for small drones, the proposed solution opens up exciting opportunities for successful autonomous flight under strict power budgets.

In order to assess the need for eNVM in energy-efficient embedded systems, we compare the proposed STT-MRAM based system over a traditional DRAM-based HBM system. We use the parameters from [5]–[7], [55]. to estimate the dissipated energy from each memory stack in three cases: (1) forward propagation, with no backpropagation, (2) forward propagation followed by learning the parameters of the last 4 layers (L4), and (3) End-to-End RL that involves forward propagation and full layer backpropagation across all the layers. Since the weights of the last three layers of the network

Layer	Processing Latency(ms)	Num. of Active PE	Power(mW)	Energy(mJ)
CONV1+ReLU+Maxpool	0.245	704	4134	1.012
CONV2+ReLU+Maxpool	1.087	960	5571	6.056
CONV3+ReLU	0.804	960	5674	4.564
CONV4+ReLU	1.28	960	5692	7.289
CONV5+ReLU+Maxpool	1.116	960	5672	6.33
FC1+ReLU	5.365	1024	6799	36.48
FC2+ReLU	1.189	1024	6800	8.091
FC3+ReLU	0.562	1024	6408	3.603
FC4+ReLU	0.28	1024	6410	1.8
FC5+ReLU	0.0005	160	1910	0.0009
total	**11.9285**	**880**	**5507**	**75.2259**

(a) Forward propagation system results

Layer	Processing Latency(ms)	Num. of Active PE	Power(mW)	Energy(mJ)	NVM Write
FC5+ReLU	0.0027	160	2094	0.006	No
FC4+ReLU	0.594	1024	6548	3.89	
FC3+ReLU	1.182	1024	6162	7.284	
FC2+ReLU	3.839	1024	5390	20.69	Yes
FC1+ReLU	29.19	1024	5390	157.3	
CONV5+ReLU+Maxpool	4.661	208	1888	8.804	
CONV4+ReLU	5.579	260	2112	11.78	
CONV3+ReLU	4.71	260	2112	9.947	
CONV2+ReLU+Maxpool	5.518	432	2850	15.73	
CONV1+ReLU+Maxpool	38.95	1024	5390	209.9	
total	**94.2257**	**644**	**3993.6**	**445.331**	

(b) Backward propagation system results

Fig. 18. Latency, power and energy of each layers in forward and backward propagation [4].

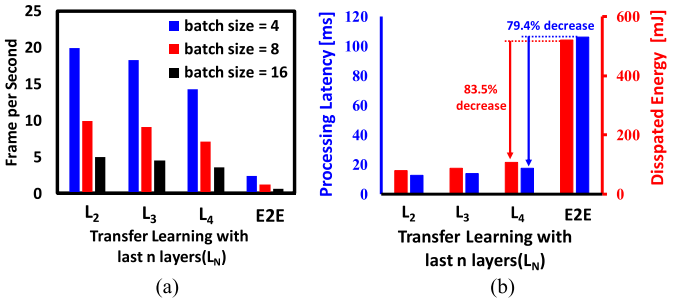


Fig. 19. (a) Maximum fps supported by different algorithms as a function of batch size. (b) Estimated processing latency and energy dissipation [4].

reside in the global buffer, the energy dissipation from the memory stack for L_2 and L_3 are same as that of forward propagation. The DRAM arrays in the DRAM-based HBM is refreshed every 64ms we consider the power cost of refreshing the entire 100MB following the JEDEC specifications.

Fig. 20 shows the energy dissipation of DRAM-based HBM and STT-MRAM based designs (off-chip) for 1000 iterations of forward propagation (no backpropagation), and training (gradient descent and backpropagation) for L_4 and E2E. The energy dissipation for the DRAM-based HBM from the figure is the sum of refresh, read/write and IO energy dissipation. Since STT-MRAM does not have refresh operation, the energy dissipation of STT-MRAM is the sum of read/write operation and IO energy dissipation. From the figure we observe that the energy dissipation from DRAM-based HBM is 2× greater than the energy dissipation from NVM in case of forward propagation. The difference in energy dissipation between DRAM-based HBM and STT-MRAM increases from L_4 to E2E since the number of refresh operations in the DRAM-based HBM is higher. This is attributed to the fact that it takes significantly longer to complete E2E compared to L_4.

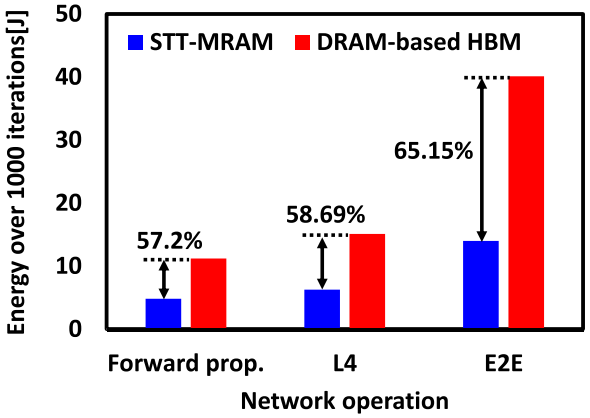


Fig. 20. Energy dissipation from DRAM-based HBM and STT-MRAM memory stack (off-chip) in case of Forward propagation, last 4 layer training (L_4) and E2E learning.

IX. Conclusion

In this paper, we present a hardware-algorithm frame-work for STT-MRAM based embedded systems for application to small drones. We show that TL followed by RL on the last few layers of a deep CNN provides comparable performance compared to an E2E RL system, while reducing latency and energy by 79.4% and 83.45% respectively.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[2] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," in *Proc. 8th Robot., Sci. Syst.* Cambridge, MA, USA: Massachusetts Institute of Technology, Jul. 2017. [Online]. Available: http://www.roboticsproceedings.org/rss13/index.html

[3] M. A. Anwar and A. Raychowdhury, "NavREn-RL: Learning to fly in real environment via end-to-end deep reinforcement learning using monocular images," *CoRR*, vol. abs/1807.08241, pp. 1–6, Jul. 2018.

[4] I. Yoon, A. Anwar, T. Rakshit, and A. Raychowdhury, "Transfer and online reinforcement learning in STT-MRAM based embedded systems for autonomous drones," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2019, pp. 1489–1494.

[5] H. Yang *et al.*, "Threshold switching selector and 1S1R integration development for 3D cross-point STT-MRAM," in *IEDM Tech. Dig.*, Dec. 2017, pp. 38.1.1–38.1.4.

[6] G. Jan *et al.*, "Demonstration of fully functional 8 Mb perpendicular STT-MRAM chips with sub-5 ns writing for non-volatile embedded memories," in *Symp. VLSI Technol. (VLSI-Technol.), Dig. Tech. Papers*, Jun. 2014, pp. 1–2.

[7] Q. Dong *et al.*, "A 1 Mb 28 nm STT-MRAM with 2.8 ns read access time at 1.2 V VDD using single-cap offset-cancelled sense amplifier and *in-situ* self-write-termination," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 480–482.

[8] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, Aug. 1988.

[9] M. van Otterlo and M. Wiering, "Reinforcement learning and Markov decision processes," in *Reinforcement Learning*. Berlin, Germany: Springer, 2012, pp. 3–42.

[10] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.

[11] A. Amravati, S. B. Nasir, S. Thangadurai, I. Yoon, and A. Raychowdhury, "A 55 nm time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 124–126.

[12] A. Amaravati, S. B. Nasir, J. Ting, I. Yoon, and A. Raychowdhury, "A 55-nm, 1.0–0.4V, 1.25-pJ/MAC time-domain mixed-signal neuromorphic accelerator with stochastic synapses for reinforcement learning in autonomous mobile robots," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 75–87, Jan. 2019.

[13] M. A. Anwar and A. Raychowdhury, "NavREn-Rl: Learning to fly in real environment via end-to-end deep reinforcement learning using monocular images," in *Proc. 25th Int. Conf. Mechatronics Mach. Vis. Pract. (M2VIP)*, Nov. 2018, pp. 1–6.

[14] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[15] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.

[16] C. J. Lin *et al.*, "45 nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell," in *IEDM Tech. Dig.*, Dec. 2009, pp. 1–4.

[17] S. Yu *et al.*, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in *IEDM Tech. Dig.*, Dec. 2016, pp. 16.2.1–16.2.4.

[18] O. Golonzka *et al.*, "MRAM as embedded non-volatile memory solution for 22FFL FinFET technology," in *IEDM Tech. Dig.*, Dec. 2018, pp. 18.1.1–18.1.4.

[19] J. Y. Wu *et al.*, "A 40 nm low-power logic compatible phase change memory technology," in *IEDM Tech. Dig.*, Dec. 2018, pp. 27.6.1–27.6.4.

[20] M. Jerry *et al.*, "Ferroelectric FET analog synapse for acceleration of deep neural network training," in *IEDM Tech. Dig.*, Dec. 2017, pp. 6.2.1–6.2.4.

[21] Q. Luo *et al.*, "8-Layers 3D vertical RRAM with excellent scalability towards storage class memory applications," in *IEDM Tech. Dig.*, Dec. 2017, pp. 2.7.1–2.7.4.

[22] K. Rho *et al.*, "A 4 Gb LPDDR2 STT-MRAM with compact $9F^2$ 1T1MTJ cell and hierarchical bitline architecture," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 396–397.

[23] F. L. Da Silva and A. H. R. Costa, "Transfer learning for multiagent reinforcement learning systems," in *Proc. 25th Int. Joint Conf. Artif. Intell. (IJCAI)*. Menlo Park, CA, USA: AAAI Press, 2016, pp. 3982–3983.

[24] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, pp. 1633–1685, Jul. 2009.

[25] M. E. Taylor and P. Stone, "Cross-domain transfer for reinforcement learning," in *Proc. 24th Int. Conf. Mach. Learn. (ICML)*, 2007, pp. 879–886.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[27] *JEDEC Standard High Bandwidth Memory(HBM) Dram Specification*, Standard JESD235B, 2015.

[28] V. D. Nguyen *et al.*, "Towards high density STT-MRAM at sub-20 nm nodes," in *Proc. Int. Symp. VLSI Technol., Syst. Appl. (VLSI-TSA)*, Apr. 2018, pp. 1–2.

[29] W. Zhao *et al.*, "High density spin-transfer torque (STT)-MRAM based on cross-point architecture," in *Proc. 4th IEEE Int. Memory Workshop*, May 2012, pp. 1–4.

[30] S. Lee *et al.*, "Highly scalable STT-MRAM with 3-dimensional cell structure using in-plane magnetic anisotropy materials," in *Proc. Symp. VLSI Technol. (VLSIT)*, Jun. 2012, pp. 65–66.

[31] Y. Huai *et al.*, "High density 3D cross-point STT-MRAM," in *Proc. IEEE Int. Memory Workshop (IMW)*, May 2018, pp. 1–4.

[32] Y. J. Lee *et al.*, "Demonstration of chip level writability, endurance and data retention of an entire 8 Mb STT-MRAM array," in *Proc. Int. Symp. VLSI Technol., Syst. Appl. (VLSI-TSA)*, Apr. 2013, pp. 1–2.

[33] J. A. O'Donnell *et al.*, "eNVM MRAM retention reliability modeling in 22FFL FinFET technology," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Mar./Apr. 2019, pp. 1–3.

[34] H. Sato *et al.*, "14 ns write speed 128 Mb density embedded STT-MRAM with endurance $>10^{10}$ and 10yrs retention@85°C using novel low damage MTJ integration process," in *IEDM Tech. Dig.*, Dec. 2018, pp. 27.2.1–27.2.4.

[35] H. Wang, D. Huang, R. Liu, C. Zhang, H. Tang, and Y. Yuan, "STREAM: Stress and Thermal Aware Reliability Management for 3D ICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, to be published.

[36] F. Beneventi, A. Bartolini, P. Vivet, and L. Benini, "Thermal analysis and interpolation techniques for a logic + wideio stacked dram test chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 4, pp. 623–636, Apr. 2016.

[37] D. Oh, C. C. P. Chen, and Y. H. Hu, "Efficient thermal simulation for 3-D IC with thermal through-silicon vias," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 11, pp. 1767–1771, Nov. 2012.
[38] S. K. Samal, S. Panth, K. Samadi, M. Saeidi, Y. Du, and S. K. Lim, "Adaptive regression-based thermal modeling and optimization for monolithic 3-D ICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1707–1720, Oct. 2016.
[39] M. Martins *et al.*, "Open cell library in 15 nm FreePDK technology," in *Proc. ACM Symp. Int. Symp. Phys. Design (ISPD)*, 2015, pp. 171–178.
[40] A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," *Solid-State Electron.*, vol. 125, pp. 25–38, Nov. 2016.
[41] J. J. Kan *et al.*, "Systematic validation of 2x nm diameter perpendicular MTJ arrays and MgO barrier for sub-10 nm embedded STT-MRAM with practically unlimited endurance," in *IEDM Tech. Dig.*, Dec. 2016, pp. 27.4.1–27.4.4.
[42] J. J. Kan *et al.*, "A study on practically unlimited endurance of STT-MRAM," *IEEE Trans. Electron Devices*, vol. 64, no. 9, pp. 3639–3646, Sep. 2017.
[43] L. Q. Luo *et al.*, "Functionality demonstration of a high-density 1.1 V self-aligned split-gate NVM cell embedded into LP 40 nm CMOS for automotive and smart card applications," in *Proc. IEEE 8th Int. Memory Workshop (IMW)*, May 2016, pp. 1–4.
[44] D. Shum *et al.*, "40 nm embedded self-aligned split-gate flash technology for high-density automotive microcontrollers," in *Proc. IEEE Int. Memory Workshop (IMW)*, May 2017, pp. 1–4.
[45] D. Kang *et al.*, "7.1 256 Gb 3b/cell V-NAND flash memory with 48 stacked WL layers," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Jan. 2016, pp. 130–131.
[46] W. Kim *et al.*, "ALD-based confined PCM with a metallic liner toward unlimited endurance," in *IEDM Tech. Dig.*, Dec. 2016, pp. 4.2.1–4.2.4.
[47] Y. J. Song *et al.*, "Highly functional and reliable 8 Mb STT-MRAM embedded in 28 nm logic," in *IEDM Tech. Dig.*, Dec. 2016, pp. 27.2.1–27.2.4.
[48] A. Chintaluri, H. Naeimi, S. Natarajan, and A. Raychowdhury, "Analysis of defects and variations in embedded spin transfer torque (STT) MRAM arrays," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 6, no. 3, pp. 319–329, Sep. 2016.
[49] D. P. O'Leary, "Systolic arrays for matrix transpose and other reorderings," *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 117–122, Jan. 1987.
[50] J. Bottleson, S. Kim, J. Andrews, P. Bindu, D. N. Murthy, and J. Jin, "clCaffe: OpenCL accelerated Caffe for convolutional neural networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 50–57.
[51] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," *CoRR*, vol. abs/1705.05065, pp. 1–14, Jul. 2017.
[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.
[53] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2009, pp. 248–255.
[54] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Univ. Cambridge, Cambridge, U.K., 1989.
[55] *Highlights of the High Bandwidth Memory (HBM) Standard*. Accessed: Aug. 2018. [Online]. Available: https://www.cs.utah.edu/thememoryforum/mike.pdf

Insik Yoon received the B.S. and M.S. degrees from Carnegie Mellon University, Pittsburgh, PA, USA, in 2009 and 2010, respectively. He is currently pursuing the Ph.D. degree with the Georgia Institute of Technology, Atlanta, GA, USA.

From 2010 to 2015, he was with Memory and Display Interface Design, TLi and SK hynix, Icheon, South Korea. His current research interests include emerging memory technologies and hardware implementation for state-of-the-art machine learning algorithms.

Malik Aqeel Anwar received the bachelor's degree in electrical engineering from the University of Engineering and Technology (UET), Lahore, Pakistan, in 2012, and the master's degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2017. He is currently pursuing the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology under the supervision of Dr. A. Raychowdhury. His research interests lie at the junction of machine learning and hardware design. He is working towards shifting machine learning (ML) from cloud to edge nodes by improving energy efficiency of current state-of-the-art ML algorithms and designing efficient DNN accelerators.

Rajiv V. Joshi (M'87–F'01) received the B.Tech. degree from IIT Bombay, Mumbai, India, the M.S. degree from MIT, Cambridge, MA, USA, and the Dr.Eng.Sc. degree from Columbia University, New York, NY, USA. He is currently a Research Staff Member with the Thomas J. Watson Research Center, IBM, Yorktown Heights, NY, USA. He has authored or coauthored more than 185 papers. He holds 58 invention plateaus, 225 U.S. patents, and more than 350 including international patents. He is a member of IBM Academy of technology. He received the Best Editor Award from the IEEE TVLSI journal, and the 2013 IEEE CAS Industrial Pioneer Award and the 2013 Mehboob Khan Award from Semiconductor Research Corporation. He was a recipient of the 2015 BMM Award. He is inducted into New Jersey Inventor Hall of Fame in 2014 along with pioneer Nikola Tesla. He served as a Distinguished Lecturer of the IEEE CAS and EDS society. He is an ISQED and World Technology Network Fellow and Distinguished Alumnus of IIT Bombay. He is on the Board of Governors for the IEEE CAS. He serves as an Associate Editor of the IEEE TVLSI. He served on committees of International Symposium Low Power Electronic Design, the IEEE VLSI Design, the IEEE CICC, the IEEE International SOI Conference, ISQED, and Advanced Metallization Program committees.

Titash Rakshit received the Ph.D. degree in electrical and computer engineering from Purdue University in 2004. His thesis involved predicting negative differential resistance at the semiconductor-molecule interfaces. He was with Intel Corporation, where he was a part of the research team that developed the industry first demonstration scaled finFET technology. Since 2014, he has been with Samsung Advanced Logic Lab focusing on future technology and systems roadmap. He is currently a Principal Engineer with the Advanced Logic Lab, Samsung Semiconductor.

Arijit Raychowdhury (SM'13) received the B.E. degree in electrical and telecommunication engineering from Jadavpur University, Kolkata, India, in 2001, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2007. From 2013 to July 2019, he was an Associate Professor and held the ON Semiconductor Junior Professorship with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. His industry experience includes 5 years as a Staff Scientist with the Circuits Research Lab, Intel Corporation, Hillsboro, OR, USA, and 1 year as an Analog Circuit Researcher with Texas Instruments, Inc. In January 2013, he joined the School of Electrical and Computer Engineering, Georgia Institute of Technology, where he is currently a Professor. He is currently the Co-Director of the Georgia Tech Quantum Alliance, Atlanta, GA, USA. He has authored or coauthored more than 170 articles in journals and refereed conferences. He holds more than 25 U.S. and international patents. His significant contributions to the semiconductor industry include the design of the world's first adaptive echo-cancellation network for integrated DSLs (TI) and embedded world-line boosting for SRAM arrays (Intel). His current research interests include low-power digital and mixed-signal circuit design, and the design of power converters, sensors, and exploring interactions of circuits with device technologies.

Dr. Raychowdhury has served on the Technical Program Committees for VLSI Symposium, CICC, DAC, ICCAD, ISLPED, and DATE. He has also taught many short courses and invited tutorials at multiple conferences, workshops, industries, and universities. He was a recipient of the IEEE/ACM Innovator under 40 Award, the Meissner Fellowship 2002, the NASA INAC Fellowship in 2004, the Intel Foundation Fellowship in 2006, the SRC Technical Excellence Award in 2005, the Dimitris N. Chorafas Award for outstanding doctoral research in 2007, the Best Thesis Award, College of Engineering, Purdue University, in 2007, the Intel Labs Technical Contribution Award in 2011, and the NSF CISE Research Initiation Initiative Award (CRII) in 2015. He and his students have received 11 best paper awards over the years. He was the Associate Editor of the IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN from 2013 to 2018 and the Editor of the *Microelectronics Journal* (Elsevier Press) from 2013 to 2017. He has also been a Guest Editor for multiple IEEE and ACM journals.