

# AVAC: A Machine Learning based Adaptive RRAM Variability-Aware Controller for Edge Devices

Shikhar Tuli\* and Shreshth Tuli†

\*Department of Electrical Engineering, Indian Institute of Technology Delhi

†Department of Computer Science and Engineering, Indian Institute of Technology Delhi

Email: {shikhartuli98, shreshthtuli}@gmail.com

**Abstract**—Recently, the Edge Computing paradigm has gained significant popularity both in industry and academia. Researchers now increasingly target to improve performance and reduce energy consumption of such devices. Some recent efforts focus on using emerging RRAM technologies for improving energy efficiency, thanks to their no leakage property and high integration density. As the complexity and dynamism of applications supported by such devices escalate, it has become difficult to maintain ideal performance by static RRAM controllers. Machine Learning provides a promising solution for this, and hence, this work focuses on extending such controllers to allow dynamic parameter updates. In this work we propose an Adaptive RRAM Variability-Aware Controller, AVAC, which periodically updates Wait Buffer and batch sizes using on-the-fly learning models and gradient ascent. AVAC allows Edge devices to adapt to different applications and their stages, to improve computation performance and reduce energy consumption. Simulations demonstrate that the proposed model can provide up to 29% increase in performance and 19% decrease in energy, compared to static controllers, using traces of real-life healthcare applications on a Raspberry-Pi based Edge deployment.

## I. INTRODUCTION

With the rapid technology burst in the Internet of Things (IoT) domain, the demands from Edge and Fog devices have been continuously increasing [1]. Edge devices are becoming more complex, expected to do more-and-more computations in order to reduce the data to be sent through the transceiver; thus reducing the bandwidth requirement, latency and energy consumption. With the eventual goal of a completely independent Edge node [2], designers are constantly making energy specifications stricter for Edge devices [1]. Further, with CMOS technology scaling to the deep sub-micron domain, the leakage power has become comparable to the logic power consumption [3]. To alleviate this problem, it is lucrative to enhance the performance of such devices and then expect them to quickly switch to the idle deep-sleep mode [4].

In that context, emerging Resistive Random Access Memory (RRAM) technologies serve as a good candidate as opposed to the traditional eFlash technologies, thanks to their non-volatile operation, zero leakage, easy co-integration with CMOS process, low programming voltage and fast switching [5], [6]. RRAMs find applications in novel system designs with non-volatile cache and universal memories [7], [8]. These new architectures can allow sufficient gains in performance and energy, much needed for upcoming Edge/Fog devices.

One such architecture is the Static RRAM Variability-Aware Controller (RRAM-VAC) proposed by S. Tuli et al. in [9]. To tackle the problem of high device-to-device and cycle-to-cycle temporal variability in RRAMs (which can be up to several decades [10]), the Static RRAM-VAC utilizes the recently proposed Write Termination (WT) circuits [11]. Thanks to their capability of dynamically detecting and stopping the programming operation once the device has switched, the Static RRAM-VAC can coalesce multiple write operations before triggering the write request. Thus, it can average out the temporal write-time variability and effectively run the system at the memory programming time distribution mean rather than the worst case tail. However, with a fixed and rigid architecture, the Static RRAM-VAC is not robust and adaptive enough to accommodate widely varied and unpredictable memory trace patterns in developing complex Edge/Fog devices.

Hence, in this work we propose a Machine Learning (ML) based approach to dynamically tune operation parameters so as to maximize performance and limit energy requirement at all times of operation. For this, we characterize different Wireless Body Sensor Node (WBSN) applications, a typical candidate for large amount of critical data which needs to be processed timely, securely and efficiently. We exploit a polynomial regression technique to model the operation parameters, namely the Wait Buffer and batch sizes with the performance and energy gains. We then apply gradient ascent method to maximize the gains dynamically, thus providing up to 94% performance gains (up to 29% more than the Static RRAM-VAC) and up to 99% energy gains (up to 19% more than the Static RRAM-VAC) on traces of real-life healthcare applications, including the HealthFog [12], [13] framework.

The rest of the paper is organized as follows. Section II presents the Static RRAM-VAC operation and the Machine Learning technique used. Section III shows the proposed AVAC architecture. Section IV describes the RRAM technology assumptions and experimental setup used for simulations. Section V compares the gains of the Adaptive over the Static RRAM-VAC. Finally, Section VI concludes the paper.

## II. BACKGROUND

### A. The Static RRAM-VAC

The Static RRAM Variability-Aware Controller (RRAM-VAC) proposed in [9] stores the write requests and locks them to form a batch. This batch is then written to an RRAM using

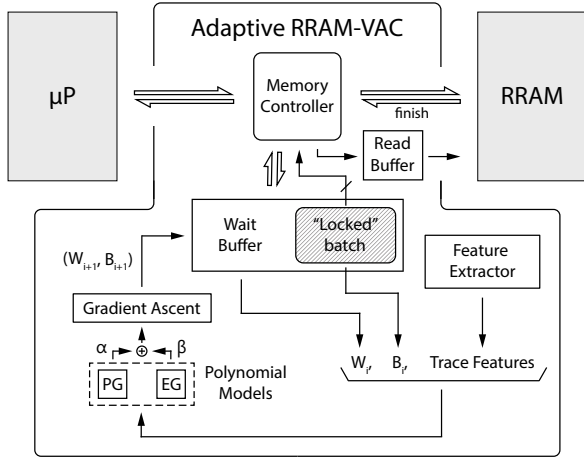


Fig. 1. Proposed Adaptive RRAM-VAC (AVAC) Block diagram

the “Write Coalescing” method - as part of a single operation, writing the next bit as-soon-as the previous one is finished [9]. This allows it to effectively interact with a synchronous processor and write to the RRAM asynchronously, further enabling the system to run at the mean of the switching-time distribution rather than at the worst-case tail.

Functionally, the Static RRAM-VAC uses a Wait Buffer and a Read Buffer. The Wait Buffer is implemented as a Binary Content Addressable Memory (BCAM) [14] of a fixed size. The locked batch is a part of the Wait Buffer and cannot catch read requests. On the other hand, the rest of the Wait Buffer can still catch both read and write requests. Read requests are first sent to the Wait Buffer. If the corresponding address is not present in the Wait Buffer, the request is issued to the RRAM. If a locked batch is in process, the read request is stored in the Read Buffer where it waits until the RRAM is available.

However, the given system has a fixed size of the Wait Buffer and batch, optimally dependent on application patterns [9]. With the increasing complexity of Edge/Fog workloads, there is a need to dynamically tune the Wait Buffer and batch sizes in order to enhance energy and performance. Moreover, these workloads correspond to real-life applications and continuously engage with unpredictable service demands and tasks. This makes prediction of optimum buffer and batch sizes difficult. Thus, in this work, we propose an on-the-fly ML based approach of dynamically tuning these sizes in order to maintain the best performance and energy characteristics.

### B. Machine Learning models

The proposed AVAC model uses polynomial regression [15] and gradient ascent [16]. The former provides a way to approximate a function with input as a multidimensional vector, characteristic to the application and the current Wait Buffer and batch sizes. We use this to approximate the expected performance and energy gains and determine a cumulative reward as described in Section III-C. We then use gradient ascent to iteratively update the Wait Buffer and batch size pair, in order to optimize this reward function.

## III. ADAPTIVE RRAM-VAC ARCHITECTURE

This section gives a functional description of AVAC.

### A. Dynamic Buffer Size

Figure 1 presents a detailed block diagram of the proposed AVAC. As explained in Section II-A, the controller fetches the read/write requests from the processor and stores them to the Wait Buffer (for a write request) or the Read Buffer (for a read request). The “locked” batches are written employing the “Write Coalescing” technique. However, in this architecture, the Wait Buffer is larger in size and we can dynamically switch off certain Write-Lines based on the requirement. This allows us to change the effective Wait Buffer size that is being used at any given time. The power-gating reduces leakage in applications where a large size is not needed, since it shuts off the current to these cells. On the other hand, for applications with very low data locality, we can optimize performance by increasing the Wait Buffer size, since that would better average the variability in write times.

Further, the whole Wait Buffer can be connected to the RRAM block via a wide bus. We can dynamically tune the batch size by varying the number of words written at a time. Higher variability ( $\sigma$ ) of the distribution requires a larger batch size. Moreover, the optimum batch size also depends on the Wait Buffer size [9].

### B. Feature Extraction

The optimum Wait Buffer and batch sizes and the possible performance and energy gains are dependent on the memory-trace features, as introduced in Section II-A. These features represent the application memory access patterns. They are extracted in the AVAC by the “Feature Extractor” as shown in Figure 1. The set of all the 8 features, with the current *Wait Buffer* size ( $W$ ) and *batch* size ( $B$ ) forms a 10 dimensional feature vector. The 8 features extracted are as follows:

- 1) Read/Write ratio: Large number of reads requires a small batch size, so that read requests do not have to wait for a long time for the batch to flush out to the RRAM. If the read locality is high, then a small Wait Buffer can also lead to performance gains (reading from the Wait Buffer is less expensive than from the RRAM block).
- 2) Read locality: As explained above, high read locality would benefit from a small Wait Buffer. If the locality is low, the application would demand a larger Wait Buffer.
- 3) Write locality: Again, high write locality would benefit from a small Wait Buffer just as the reads above.
- 4) Mean Read burst size: For a series of small read bursts, the Wait Buffer will have to wait frequently for the Read Buffer to clear out, thus not allowing it to flush out its write requests so as to catch the next write instruction.
- 5) Mean Write burst size: While the batch is being written to memory, the next read request that cannot be processed inside the Wait Buffer is stored in the Read Buffer and the processor is stalled, compromising performance.
- 6) Mean Read repetition: The read locality only targets the variance in the read request addresses. The peak of

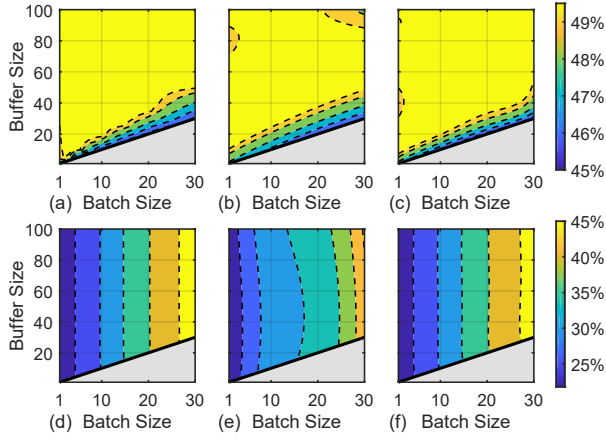


Fig. 2. Performance gains (a) actual, and model with degree - (b) 3 and (c) 5. Energy gains (d) actual, and model with degree - (e) 3 and (f) 5.

the distribution can be represented by the mean of the repetition for each read address.

- 7) Mean Write repetition: Similar to the case of read repetition, higher write repetition would result in higher energy gains by accesses from the Wait Buffer.
- 8) Variation in bit-changes for writes: By this feature, we mean the number of changes in bit written to a particular address (0 to 1 or 1 to 0). The Write Coalescing strategy works best if the variability in writes can be averaged out. However, for example, if the writes to addresses are of the form 0x00000000 to 0x0000000F, then the LSB will always have higher write times than the other bits, thus reducing the gains of a higher batch size.

### C. Polynomial model and parameter optimization

The complete feature vector, calculated for a fixed number of memory accesses in 10 dimensions, is given to two polynomial-fit based models: Performance Gains (PG) and Energy Gains (EG). We empirically find 1000 to be the best interval size ( $S_I$ ), and propose to test with other and dynamic  $S_I$  values as part of future work. Each model is an approximator that maps the 10 dimensional feature vector ( $fv$ ) to the corresponding performance and energy gains ( $pg$  and  $eg$ ). At training time, we generate multiple data-points  $\{fv^{(j)}, pg^{(j)}, eg^{(j)}\}_{j=1}^N$  for different applications with buffer size  $\in [1, 120]$  and batch size  $\in [1, 80]$ . This training data is then used to perform polynomial regression and generate model parameters for PG and EG. The applications were chosen to cover a large space of feature vectors.

At testing time, for every memory trace of last 1000 accesses at time  $t_i$ , the trace feature vector is coupled with the current (at time  $t_i$ ) buffer size ( $W_i$ ) and batch size ( $B_i$ ) to form a vector of size 10 which is sent to the two polynomial models that output  $pg_i$  and  $eg_i$ . These are then used to calculate the reward ( $r_i$ ) as convex combination  $\alpha \cdot pg_i + \beta \cdot eg_i$  where the weights  $\alpha$  and  $\beta$  ( $\alpha + \beta = 1$ ) can be changed based on user requirements targeting either optimal performance or energy (experiments in this work use  $\alpha = 0.1$  and  $\beta = 0.9$  to prioritize

energy reduction). This whole pipeline is run multiple times by changing buffer and batch sizes to achieve maximum reward using gradient ascent. For the current model, we used learning rate as 0.01 with 0.9 momentum. This gives the  $(W_{i+1}, B_{i+1})$  tuple which maximizes the reward (keeping area minimum) and is used for the next interval till  $t_{i+1}$ . Figure 2 shows actual performance and energy gains with polynomial models of degree 3 and 5. With higher degree, outputs are closer to actual values but require more model parameters (AVAC uses degree 5 models). Finally, the Wait Buffer and batch sizes were fixed to 80 and 10 respectively for the Static RRAM-VAC [9].

## IV. EXPERIMENTAL SETUP

This section discusses the basic setup, energy and speed assumptions, with biomedical applications and generic benchmarks. The simulations were performed on MATLAB.

### A. Energy and Speed Assumptions

For a programming voltage of 1V to 1.5V, programming time of a few tens of nano-seconds can be achieved [5]. Hence, we have assumed a 50 ns worst-case programming time at a programming voltage of 1V. 100  $\mu$ A programming current has been assumed for a high ratio of resistance between the High Resistance State (HRS) and the Low Resistance State (LRS) and for several years of memory lifetime [5]. We take a normal distribution of programming time with the mean ( $\mu$ ) at 25 ns and a variance ( $\sigma$ ) of 5 ns [9]. The AVAC model can be further extended to account for ageing and corresponding variation in the distribution. As explained in [9], energy characterization is done for every bit as an integral ( $V \int I \cdot dt$ ) taking into account the shift in the current ( $I$ ) for every stochastic transition (HRS to LRS or vice-versa). WT circuit switching detection time is taken to be 1 ns [9]. The read energy is considered as 1 pJ per bit [17]. Scaling down CMOS technology has led to increase in leakage power, primary concern in low-power circuits. The leakage current (for 1V programming voltage) can go as high as 15nA per bit cell (and even higher in some cases) in the current deep-submicron nodes [18]. This leads to a leakage gain of 480nW per word-line of 32 bits (switching them off dynamically when not needed) in the proposed approach. Implementation of AVAC models described in Sections III-B and III-C on Artix-7 FPGA shows that it has 0.278% energy overhead compared to a 4GB RRAM, which is negligible [19].

### B. Applications and Benchmarks

Experiments were performed on different applications relevant to Edge devices in the biomedical domain. These include Compressed Sensing (CS) for an Electro-Cardiogram (ECG) signal, Feature Extraction (FE) and Decision Tree (DT) in Epilepsy seizure detection algorithm, with Matrix Multiplication (MM) and Convolution (Conv) as two kernels [20], [21] as used in [9]. DT\_C is the DT application post-processed with an L1 cache of 16 words. We also use Sysbench CPU [22] and Apache [23] to further increase the diversity of applications. All benchmarks traces were extracted on Raspberry-Pi 4 and used to fit the polynomial models PG and EG.

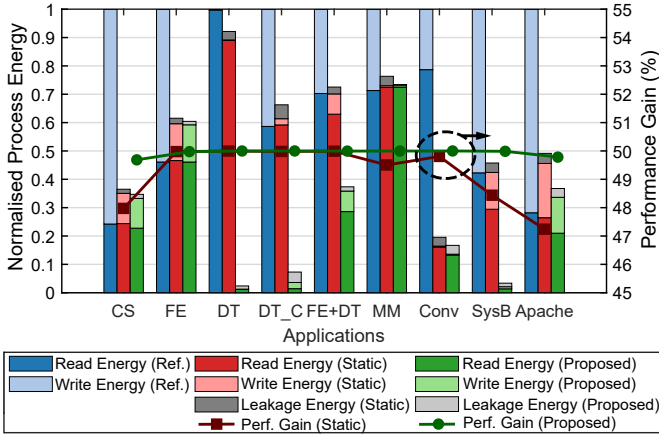


Fig. 3. Energy and Performance comparison for different applications

## V. EXPERIMENTAL RESULTS

Figure 3 compares the energy and performance gains for the Static and the Adaptive RRAM-VAC with the reference case not using the RRAM-VAC. These gains can be explained by comparing the model parameter optima, i.e the optimal values of the tuple  $(W, B)$  for different applications, with that of the Static RRAM-VAC - (80, 10). For the applications CS and FE, the parameter optima ((80, 60) and (70, 50) respectively) were quite close to that of the Static RRAM-VAC, giving only about 2% more gains in energy compared to the Static case. The CS and FE have low locality of addresses, and a low read/write ratio. Since the parameters for Static RRAM-VAC were decided based on writes to random addresses [9] (implying a low locality and zero read/write ratio) these memory traces are quite close to the assumed one in the Static RRAM-VAC. Still, CS gains a little in performance owing to a larger batch size (averaging the variability in write times). However, MM and Conv show little dependence of the parameters on the gains, with their respective optima at (10, 10) and (80, 80). This is due to the fact that both have balanced read/write ratios. In MM, AVAC reduces leakage power by reducing the Wait Buffer size as locality is low, since a larger Wait Buffer does not provide significant advantage. On the other hand, Conv has high gains both in the Static and Adaptive RRAM-VAC and benefits from a large Wait Buffer to reduce energy of read and write accesses.

AVAC provides high gains in DT, DT\_C and FE+DT, with their respective model optima at - (30,30), (60,60) and (50,50). DT shows maximum improvement in energy gains-89% higher than the Static case. With otherwise low locality but high address repetition, these applications perform better with a smaller Wait Buffer. This can be explained by high read/write burst size and high address repetition. Sysbench shows high gains with the model optimum at (30,30) due to high read/write locality. On the other hand, Apache has similar optimum at (70,50) as the Static case, due to low read/write locality. Both benefit with a larger batch size, providing 3% higher performance gains compared to the Static case.

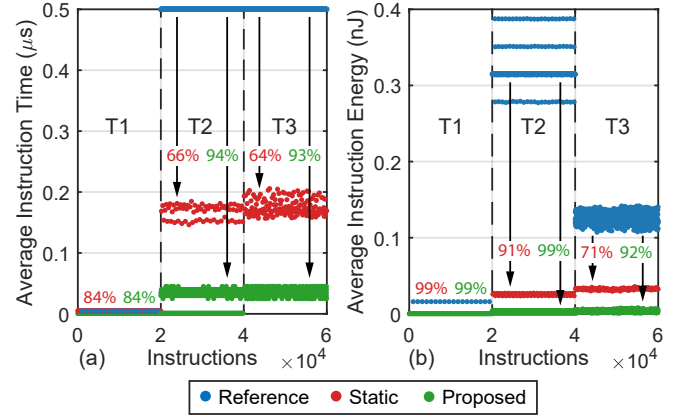


Fig. 4. Transient simulations for Static and Adaptive RRAM-VAC for HealthFog application (a) Performance Gains and (b) Energy Gains

To demonstrate efficacy of AVAC, we further show the performance and energy gains for a real-life Edge application called HealthFog [12] which provides high accuracy healthcare services using ensemble deep learning. Figure 4 shows the application running in 3 major stages: (T1) the program performs large number of ECG read operations and shares data in real-time with sensors and actuators; (T2) the program performs task scheduling and migration decisions for minimum service-level-agreement violations [2]; (T3) the program utilizes ensemble deep learning based methods to evaluate the data and generate results like health analysis and automated prescription generation. Gains in static and adaptive cases are same in T1 due to only-read operations. T2 has large number of convolution operations giving high energy gains in both the Static and Adaptive cases, compared to the reference. Further, T3 has many MM-like operations and also bootstrapping processes similar to those in the DT, leading to higher gains (29% performance and 19% energy). Also, Figure 4 highlights how the proposed ML-based model can adapt to shifts in memory access patterns to instantly enhance gains.

## VI. CONCLUSIONS

In this work, we proposed the Adaptive RRAM Variability-Aware Controller (AVAC), which uses Machine Learning (ML) techniques to dynamically configure the Wait Buffer and the batch sizes. This not only mitigates the device-to-device variability in RRAMs, but also further optimizes performance and energy dependent on the memory access patterns. Gains were simulated for different applications in the Edge/Fog environment, and compared to the Static RRAM-VAC with fixed Wait Buffer and batch sizes. With the considered RRAM technology, AVAC provides up to 94% gains in performance (up to 29% more than the Static case) and up to 99% gains in energy (up to 19% higher than the Static case). The model can dynamically tune operation parameters in complex and varied Edge and Fog applications to maintain the best performance and energy at all times. Other ML models, with dynamic interval sizing can be investigated as part of future work.

## REFERENCES

- [1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [2] S. S. Gill, S. Tuli, M. Xu, I. Singh, K. V. Singh, D. Lindsay, S. Tuli, D. Smirnova, M. Singh, U. Jain, H. Pervaiz, B. Sehgal, S. S. Kaila, S. Misra, M. S. Aslanpour, H. Mehta, V. Stankovski, and P. Garraghan, "Transformative effects of IoT, Blockchain and Artificial Intelligence on Cloud Computing: Evolution, vision, trends and open challenges," *Internet of Things*, vol. 8, p. 100118, 2019.
- [3] International Technology Roadmap for Semiconductors by Semiconductor Industry Association, 2011. [Online]. Available: <http://www.itrs2.net/2011-itrs.html>
- [4] R. Braojos, D. Atienza, M. M. S. Aly, T. F. Wu, H.-S. P. Wong, S. Mitra, and G. Ansaloni, "Nano-engineered architectures for ultra-low power wireless body sensor nodes," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Code-sign and System Synthesis*, ser. CODES '16. New York, NY, USA: ACM, 2016, pp. 23:1–23:10.
- [5] E. Vianello, O. Thomas, M. Harrand, S. Onkaraiah, T. Cabout, B. Traor, T. Diokh, H. Oucheikh, L. Perniola, G. Molas, P. Blaise, J. F. Nodin, E. Jalaguier, and B. De Salvo, "Back-end 3D integration of HfO<sub>2</sub>-based RRAMs for low-voltage advanced IC digital design," in *Proceedings of 2013 International Conference on IC Design Technology (ICICDT)*, May 2013, pp. 235–238.
- [6] H.-S. P. Wong, H. Lee, S. Yu, Y. Chen, Y. Wu, P. Chen, B. Lee, F. T. Chen, and M. Tsai, "MetalOxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, June 2012.
- [7] S. Senni, L. Torres, G. Sassatelli, A. Gamatie, and B. Mussard, "Emerging Non-volatile Memory Technologies Exploration Flow for Processor Architecture," in *2015 IEEE Computer Society Annual Symposium on VLSI*, July 2015, pp. 460–460.
- [8] W. Zhao, L. Torres, L. V. Cargnini, R. M. Brum, Y. Zhang, Y. Guilleminet, G. Sassatelli, Y. Lakys, J.-O. Klein, D. Etienne, D. Ravelosona, and C. Chappert, "High Performance SoC Design Using Magnetic Logic and Memory," in *VLSI-SoC: Advanced Research for Systems on Chip*, S. Mir, C.-Y. Tsui, R. Reis, and O. C. S. Choy, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 10–33.
- [9] S. Tuli, M. A. Rios, A. S. J. Levisse, and D. Atienza Alonso, "RRAM-VAC: A Variability-Aware Controller for RRAM-based Memory Architectures," *Proceedings of the 25th Asia and South Pacific Design Automation Conference ASP-DAC*, 2020.
- [10] G. Sassine, C. Nail, L. Tillie, D. A. Robayo, A. Levisse, C. Cagli, K. E. Hajjam, J. Nodin, E. Vianello, M. Bernard, G. Molas, and E. Nowak, "Sub-pJ consumption and short latency time in RRAM arrays for high endurance applications," in *2018 IEEE International Reliability Physics Symposium (IRPS)*, March 2018, pp. P-MY.2–1–P-MY.2–5.
- [11] M. Alayan, E. Muhr, A. Levisse, M. Bocquet, M. Moreau, E. Nowak, G. Molas, E. Vianello, and J. M. Portal, "Switching Event Detection and Self-Termination Programming Circuit for Energy Efficient ReRAM Memory Arrays," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 5, pp. 748–752, May 2019.
- [12] S. Tuli, N. Basumatary, S. S. Gill, M. Kahani, R. C. Arya, G. S. Wander, and R. Buyya, "HealthFog: An Ensemble Deep Learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in Integrated IoT and Fog Computing Environments," *Future Generation Computer Systems*, 2019.
- [13] S. Tuli, R. Mahmud, S. Tuli, and R. Buyya, "FogBus: A Blockchain-based Lightweight Framework for Edge and Fog Computing," *Journal of Systems and Software*, vol. 154, pp. 22 – 36, 2019.
- [14] A. Agarwal, S. Hsu, S. Mathew, M. Anders, H. Kaul, F. Sheikh, and R. Krishnamurthy, "A 128x128b high-speed wide-and match-line content addressable memory in 32nm CMOS," in *2011 Proceedings of the ESSCIRC (ESSCIRC)*, Sep. 2011, pp. 83–86.
- [15] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2012, vol. 821.
- [16] S. Bubeck *et al.*, "Convex optimization: Algorithms and complexity," *Foundations and Trends® in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.
- [17] P. Jain, U. Arslan, M. Sekhar, B. C. Lin, L. Wei, T. Sahu, J. Alzate-vinasco, A. Vangapaty, M. Meterelliyo, N. Strutt, A. B. Chen, P. Hentges, P. A. Quintero, C. Connor, O. Golonzka, K. Fischer, and F. Hamzaoglu, "13.2 A 3.6Mb 10.1Mb/mm<sup>2</sup> Embedded Non-Volatile ReRAM Macro in 22nm FinFET Technology with Adaptive Forming/Set/Reset Schemes Yielding Down to 0.5V with Sensing Time of 5ns at 0.7V," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, Feb 2019, pp. 212–214.
- [18] D. Bhattacharya, A. N. Bhoj, and N. K. Jha, "Design of Efficient Content Addressable Memories in High-Performance FinFET Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 5, pp. 963–967, May 2015.
- [19] Y. Liu, Z. Wang, A. Lee, F. Su, C. Lo, Z. Yuan, C. Lin, Q. Wei, Y. Wang, Y. King, C. Lin, P. Khalili, K. Wang, M. Chang, and H. Yang, "4.7 A 65nm ReRAM-enabled nonvolatile processor with 6 reduction in restore time and 4 higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 84–86.
- [20] A. Vasudevan, A. Anderson, and D. Gregg, "Parallel Multi Channel convolution using General Matrix Multiplication," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, July 2017, pp. 19–24.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105.
- [22] A. Kopytov, (2004) SysBench: a system performance benchmark. [Online]. Available: <http://sysbench.sourceforge.net/>
- [23] Apache HTTP server benchmarking tool. [Online]. Available: <https://httpd.apache.org/docs/2.4/programs/ab.html>