

EMBRACING VISUAL EXPERIENCE AND DATA KNOWLEDGE: EFFICIENT  
EMBEDDED MEMORY DESIGN FOR BIG VIDEOS AND DEEP LEARNING

A Thesis  
Submitted to the Graduate Faculty  
of the  
North Dakota State University  
of Agriculture and Applied Science

By

Jonathon David Edstrom

In Partial Fulfillment of the Requirements  
for the Degree of  
DOCTOR OF PHILOSOPHY

Major Department:  
Electrical and Computer Engineering

May 2019

Fargo, North Dakota

North Dakota State University  
Graduate School

---

**Title**

EMBRACING VISUAL EXPERIENCE AND DATA KNOWLEDGE:  
EFFICIENT EMBEDDED MEMORY DESIGN FOR BIG VIDEOS AND  
DEEP LEARNING

---

**By**

Jonathon David Edstrom

---

The Supervisory Committee certifies that this *disquisition* complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**DOCTOR OF PHILOSOPHY**

SUPERVISORY COMMITTEE:

Scott Smith

---

Chair

Na Gong

---

Sudarshan Srinivasan

---

Yiwen Xu

---

Mijia Yang

---

Approved:

5/3/19

---

Date

Benjamin D. Braaten

---

Department Chair

## ABSTRACT

Energy efficient memory designs are becoming increasingly important, especially for applications related to mobile video technology and machine learning. The growing popularity of smart phones, tablets and other mobile devices has created an exponential demand for video applications in today's society. When mobile devices display video, the embedded video memory within the device consumes a large amount of the total system power. This issue has created the need to introduce power-quality tradeoff techniques for enabling good quality video output, while simultaneously enabling power consumption reduction. Similarly, power efficiency issues have arisen within the area of machine learning, especially with applications requiring large and fast computation, such as neural networks. Using the accumulated data knowledge from various machine learning applications, there is now the potential to create more intelligent memory with the capability for optimized trade-off between energy efficiency, area overhead, and classification accuracy on the learning systems. In this dissertation, a review of recently completed works involving video and machine learning memories will be covered. Based on the collected results from a variety of different methods, including: subjective trials, discovered data-mining patterns, software simulations, and hardware power and performance tests, the presented memories provide novel ways to significantly enhance power efficiency for future memory devices. An overview of related works, especially the relevant state-of-the-art research, will be referenced for comparison in order to produce memory design methodologies that exhibit optimal quality, low implementation overhead, and maximum power efficiency.

## ACKNOWLEDGEMENTS

I would like to thank Dr. Na Gong for her time, guidance, and knowledge in completing all the works presented in this dissertation. I would also like to thank my other committee members, Dr. Scott Smith, Dr. Yiwen Xu, Dr. Sudarshan Srinivasan, and Dr. Mijia Yang for their feedback and assistance in presentations, papers and other advice given during my doctoral study. I am thankful for all the help provided by the other members of my lab, especially Yifu Gong, Dongliang Chen, and Hritom Das. Their help with preparing experiments, calculating simulation results, and verifying designs was paramount in completing the necessary work for conferences, journals, and the completion of my doctoral requirements.

I am grateful to Dr. Mark McCourt and the Department of Psychology at NDSU for allowing us to use their facility for psychophysical tests. I would specifically like to thank Ganesh Padmanabhan and Enrique Alvarez Vazquez for the assistance in analyzing video samples, preparing a mobile application testing environment for subjective trials, and preparing results for further discussion. Without their help, the process for obtaining results would have taken much longer and been much more difficult.

Last, I would like to recognize and thank the National Science Foundation and ND EPSCoR for the financial support that made the included research possible. I am grateful to NDSU's Center for Computationally Assisted Science and Technology (CCAST) for allowing me access to their high power computing servers for computing simulation results; without this, a lot of my research relating to data-mining and machine learning would not have been possible.

In reference to IEEE copyrighted material, which is used with permission in this dissertation, the IEEE do not endorse any of North Dakota State University's products or services. Internal or personal use of this material is permitted.



## **DEDICATION**

To my parents, Fred and Terri, who have always been there to support me in everything I do.

## TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
DEDICATION.....	v
LIST OF TABLES.....	x
LIST OF FIGURES.....	xi
LIST OF ABBREVIATIONS.....	xiii
CHAPTER 1. INTRODUCTION.....	1
Motivation for Video and Machine Learning Memory Optimization.....	1
Memory Design for Video Applications.....	2
Memory Design for Deep Learning Applications.....	5
Design Trade-offs and Evaluation.....	6
CHAPTER 2. DATA-PATTERN ENABLED SELF-RECOVERY LOW-POWER STORAGE SYSTEM FOR BIG VIDEO DATA.....	8
Embedded Memory Failure Analysis at Near-Threshold Voltage.....	8
Data Pattern Investigation for Self-Recovery.....	10
Rule Mining Enabled Horizontal Association.....	10
Vertical Correlation.....	13
Optimal Data Patterns for Self-Recovery.....	15
Recovery Failure Caused by Double Faults in Data Patterns.....	17
DPSR Hardware Implementation.....	17
Evaluation Methodology and Results.....	20
Performance.....	20
Layout.....	20
Power Efficiency.....	21
Video Output Quality Analysis.....	22

Comparison with Prior Work .....	26
DPSR Concluding Remarks .....	27
<b>CHAPTER 3. CONTENT-ADAPTIVE MEMORY FOR VIEWER-AWARE ENERGY-QUALITY SCALABLE MOBILE VIDEO SYSTEMS .....</b>	<b>28</b>
Influence of Video Content on Viewer Experience .....	28
Mobile Video Memory System .....	28
Influence of Video Content on Viewer Experience in the Presence of Hardware Noise.....	30
Modeling Process .....	35
Subjective Testing Procedure for Data Collection .....	36
Modeling Process .....	36
Quality Optimized Bit Truncation Design .....	40
Quality Optimized Bit Truncation.....	40
Content-Adaptation Video Memory Design .....	42
Experimental Results.....	43
Speed .....	44
Layout.....	44
Power Savings .....	45
Video Quality .....	46
Context-Aware Memory Concluding Remarks.....	51
<b>CHAPTER 4. DATA-DRIVEN INTELLIGENT EFFICIENT SYNAPTIC STORAGE FOR DEEP LEARNING .....</b>	<b>52</b>
Synaptic Storage and Memory Failure Overview .....	52
Synaptic Storage in Artificial Neural Networks.....	53
Impact of Synaptic Memory Failures on ANN Classification Accuracy .....	54
Data Characteristics of Synaptic Storage .....	57

Data Contribution Characteristics .....	57
Data Switching Characteristics.....	59
Data Association / Correlation Characteristics.....	60
Proposed Data-Driven Synaptic Memory .....	61
Implementation.....	61
Results .....	62
Data-Driven Synaptic Memory Concluding Remarks .....	64
<b>CHAPTER 5. ENABLING ENERGY-EFFICIENT DIFFERENTIALLY PRIVATE EDGE INFERENCE FOR DEEP LEARNING.....</b>	<b>66</b>
Learning with Differential Privacy .....	66
Why do we need Deep Learning with Privacy? .....	66
Differentially Private Deep Learning and State of the Art.....	68
Impact of Memory Failures in Differentially Private Deep Learning Systems .....	71
Impact of Image Quality on Classification Accuracy .....	71
Impact of Dataset Memory in Edge Devices.....	73
Impact of Hardware on Privacy/Accuracy Trade-off.....	74
Optimization Model based Memory Design .....	75
Embedded Memory Design for Deep Learning .....	76
Optimized Memory Design .....	78
Power Consumption .....	79
Dataset Quality and Accuracy .....	79
Accuracy at Different Privacy Levels .....	81
Differential Private Edge Inference Memory Concluding Remarks .....	82
<b>CHAPTER 6. CONCLUSIONS AND FUTURE WORK.....</b>	<b>83</b>
<b>REFERENCES .....</b>	<b>85</b>
<b>APPENDIX A. YOUTUBE-8M VIDEO DOWNLOAD SCRIPT .....</b>	<b>94</b>

APPENDIX B. YOUTUBE-8M VIDEO CLIPPED TIMING SCRIPT .....	95
APPENDIX C. MACROBLOCK ANALYSIS AND TRUNCATION PROGRAM.....	96
APPENDIX D. SYNAPTIC STORAGE FOR DEEP LEARNING MODEL CODE.....	104
APPENDIX E. MNIST BIT FAULT INJECTION PROGRAM .....	108

## LIST OF TABLES

<u>Table</u>	<u>Page</u>
1. Fault probability in a 32-bit SRAM word ( $10^9$ Monte Carlo simulations) .....	10
2. Discovered association rules. Bit 1 (i.e. Y1, Cb1, Cr1) is the MSB .....	12
3. Vertical correlation probabilities .....	14
4. Optimal data patterns from 25 YouTube-8M videos .....	16
5. DPSR recovery failure rates .....	17
6. Video PSNR metric comparison .....	25
7. Video SSIM metric comparison .....	25
8. DPSR comparison with prior works on low power SRAM .....	27
9. Video memories and their functionality .....	29
10. Results of videos with different LSBs truncated in different memories .....	31
11. PSNR of different videos with bit truncated applied .....	33
12. Results of ordinal logistic regression .....	39
13. Probability mass function for random variable $Y$ .....	41
14. ANN architecture and configurations .....	58
15. Synaptic storage comparison with existing 8T+6T hybrid design .....	65
16. 6T and 8T bit cell design options for 45nm technology at 0.5V .....	78
17. Optimal design results and comparison .....	79
18. Power consumption of optimized 45nm memory design at 0.5V .....	79
19. Dataset quality and accuracy for MNIST and Fashion .....	80
20. Impact of privacy level on test accuracy .....	82

## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. Trade-off between hardware evaluation metrics .....	7
2. Error maps in SRAM array at 0.5V. (a) Error rate: $10^{-3}$ and (b) error rate: $10^{-2}$ .....	9
3. 2D data-pattern enabled self-correction and data pattern analysis dataset.....	11
4. Proposed DPSR with data self-recovery ability .....	18
5. DPSR layout design.....	20
6. Power consumption of different memory operations .....	21
7. Video output using different video storage techniques .....	23
8. Mobile video memory architecture. ....	29
9. Plain macroblock visualization and video output comparison (white = plain) .....	35
10. Acceptable truncated bits based on subjective feedback .....	37
11. Developed decision tree model for bit truncation.....	38
12. Average PSNR values using two different truncation techniques .....	42
13. Content-adaptive video memory .....	43
14. Timing diagram (DATA7: MSB; DATA0: LSB).....	44
15. Physical layout design.....	45
16. Power Savings.....	46
17. Psychological experiment setup.....	47
18. Video quality testing results using the decision tree model.....	49
19. Output quality of video with tag <i>wF6lvdXXwc4</i> .....	50
20. (a) ANN architecture; (b) Single neuron with synaptic weights.....	53
21. (a) Schematic and (b) layout design of 6T and 8T SRAM bit cells.....	55
22. 45nm SRAM bit cell failure rates based on $V_{dd}$ voltage scaling .....	56
23. IEEE 754 Single Precision Floating Point Representation .....	58

24. Influence of synaptic weight bit position on ANN classification accuracy .....	59
25. MNIST average bit switching percentage of each bit position .....	59
26. Offline data-mining assisted synaptic data relationships study .....	60
27. Data-driven efficient synaptic storage .....	62
28. Layout of the proposed synaptic memory in 45 nm technology.....	63
29. Power consumption of synaptic storage.....	64
30. Proposed deep learning system with energy-efficiency/privacy/accuracy .....	70
31. Differentially private convolutional neural network used for analysis.....	71
32. Influence of dataset quality on test accuracy (MNIST dataset results).....	72
33. Impact of errors on privacy/accuracy. MSBs protected: (a) None (b) 2.....	73
34. Impact of memory failure rate on the accuracy of the learning system .....	74
35. SRAM bit cells. Minimum sized 45nm schematic and layout: (a) 6T (b) 8T.....	77
36. Verification of errors on privacy/accuracy. MSBs protected: (a) None (b) 2.....	81



## LIST OF ABBREVIATIONS

CMOS	Complementary Metal-Oxide-Semiconductor
SRAM	Static Random Access Memory
ECC	Error Correcting Code
BIST	Built-In Self-Test
LSB	Least Significant Bit
MSB	Most Significant Bit
IoT	Internet of Things
PSNR	Peak Signal-To-Noise Ratio
NMOS	N-Type Metal-Oxide-Semiconductor
PMOS	P-Type Metal-Oxide-Semiconductor
$V_{dd}$	Supply Voltage
RBL	Readout Bit Line
MUX	Multiplexer
POST	Power-On Self-Test
RDF	Random Dopant Fluctuation
URL	Uniform Resource Locator
DPSR	Data Pattern Self-Recovery
MSE	Mean Squared Error
SSIM	Structural Similarity
ANN	Artificial Neural Network
CNN	Convolutional Neural Network

## CHAPTER 1. INTRODUCTION

Recently, research and development of energy-efficient memory for general use or application specific designs has become of great interest [1]. The need for devices that are capable of saving power intrinsically, while maintaining a robust, minimal failure operation is becoming more and more necessary with the growing use of portable electronics [2]. The detailed description of multiple techniques to allow for power savings, including partially disabling circuitry or minimizing supply voltage and the use of self-correction techniques to mitigate errors and provide a high quality output, will be introduced in the different chapters within this dissertation. This chapter contains a brief introduction describing the importance of the presented designs and related works. The main two topics that will be described in detail in this dissertation involve video and deep learning applications.

### **Motivation for Video and Machine Learning Memory Optimization**

According to market research, by the year 2020, the total amount of data that will have been created, transmitted, and replicated, will be as large as 40ZB (Zettabyte, or  $10^{21}$  bytes) [3, 4]; and more than half of that data will be video data [5]. Video streaming is currently one of the most energy-intensive applications on mobile devices, with frequent memory accesses contributing to over 30% of the total power consumption [6, 7]. The frequent memory accesses required for these types of streaming applications lead to shorter battery life, which is becoming one of the largest contributors to user dissatisfaction [5]. One other notable concern is the increased memory footprint in mobile video systems. With recent advances to high definition video formats, the video decoder memory can occupy more than 65% of the total silicon area [8, 9]. As the silicon area increases, the total cost to manufacture the memory increases; therefore, designing memory with minimal area overhead is of paramount importance to the end user.

Similar to video memory design, the growing demand for computing and memory resources in machine learning applications has created a need for efficient hardware schemes to enable real-time adaptation. For example, deep learning systems such as neural networks require extensive computational time and dissipate a large amount of power to calculate a high accuracy trained model [10]. In particular, the training process needs continuous model updating and requires intensive memory access. For these types of designs, the on-chip SRAM occupies more than 56% of the silicon area and contributes to over 60% of the power consumption of the entire system [11]. Consequently, improving the energy efficiency of SRAM with low area overhead is vital to support future deep learning systems.

### **Memory Design for Video Applications**

Video applications have been shown to inherently possess error resiliency to some extent [12]. This error resiliency enables video applications to be redesigned using approximate computing methods for power savings. The video memory designs described within the chapters of this dissertation deal with mobile video applications in particular; but these designs can be adapted for use in devices that are a part of other application types as well. The designed memories use CMOS SRAM technology, but the methodologies used to incorporate power savings and data correction may prove useful in future technologies as well.

Voltage scaling techniques have been widely applied to reduce the power consumption of memory systems. Researchers have shown that SRAM achieves maximum efficiency at near-threshold voltage [13]. However, as voltage scales, SRAMs are susceptible to failure due to significant process variation. Various techniques have been developed to correct or eliminate these memory failures as voltage is scaled. Traditional low power memory techniques can be divided into three general categories: (i) assist schemes, such as adjustment of cell voltage [14]

and boosted word-line voltage [15]; (ii) large bit cells, such as upsized 6T cell [16], asymmetric 7T cell [17], single-ended read-decoupled 8T cells [18], read-disturb-free 9T [19], and bit-interleaving 12T cells [20]; and (iii) error correction techniques spanning from the use of error correction codes [21] to data remapping [22]. Unfortunately, almost all existing solutions require considerable implementation overhead, as high as 50-100% of the original memory design.

Recently, a new branch of low voltage embedded memory techniques have been developed to embrace the memory faults, instead of avoiding the faults (i.e. assist schemes or more than 6T bit cells) or correcting the faults (e.g. ECC). Those techniques aim to mitigate the impact of memory faults by minimizing the magnitude of the error due to a faulty cell, based on the determined memory fault positions from runtime testing (e.g. BIST). Those techniques are referred to as fault-position aware mitigation techniques. For example, in [23], a shifting technique was developed to store the LSBs in faulty bit cells, which may lead to a tolerable output quality. In [13], a squeezing technique was presented to compress zeros and store them in less memory space, thereby avoiding memory failures at low voltages. However, based on the predetermined memory fault positions, the existing techniques still involve complex operation (e.g. shifting value calculations and storage) and the overhead incurred is still significant (e.g. 65% in [23]). Several recent efforts have also investigated application resilience of videos to approximations with “good enough” output and additional power savings. [24] presented a hybrid 6T+8T SRAM to achieve quality-power optimization. In [16], a heterogeneous sizing scheme is presented to reduce the failure probability of conventional 6T bit cells. In [25], the correlation between the MSBs of video data was utilized to design a hybrid 8T+10T memory for power savings.

To address the storage challenge of videos as well as other big data, leveraging the assets of big data to extract useful knowledge and actionable information for hardware design is proposed in chapter 2. Today's big data applications, including videos, have three common data characteristics [26]: redundant inputs, multiple acceptable outputs, and statistical computations. Those intrinsic characteristics provide substantial opportunities for data relationship discovery and pattern identification. This in turn will enable a new dimension for hardware design space and bring exciting innovation opportunities for multi-dimensional innovations in circuits and systems. Based on this, an extension of the work done in [27] proposes an efficient SRAM design technique for big video data. The design is presented in chapter 2 with negligible area overhead (7.94%) and performance penalty.

Viewer's experience is one other important design concern that hardware engineers need to consider when designing circuitry such as the memory used for storing video. Various studies have displayed the impact of contextual influences such as illuminance levels, where an increased amount of ambient luminance allows for a larger amount of error to be introduced by voltage reduction techniques without noticeable degradation to the viewers. Two low power techniques in particular, voltage scaling and bit truncation, have been explored [28, 29, 30] and achieve similar PSNR values. However, the video quality degradation caused by bit truncation is much less noticeable than that of the voltage scaling technique for viewers.

Based on previous works [28, 29, 30] and the novel introduction of video content characteristics, chapter 3 proposes a video memory design that uses the viewer's experience to enable video content-adaptive functionality with dynamic energy-quality trade-off.

## Memory Design for Deep Learning Applications

Traditional low power memories are implemented to accommodate a large amount of data utilizing more than 6T bit cells or assist schemes which usually come with significant implementation cost (e.g. silicon area, performance overhead). Although such overhead might be acceptable in general-purpose systems, they are not sufficient to satisfy the storage need for deep learning applications. Recently in [31], a deep learning specific hybrid 8T-6T synaptic storage was presented where varying number of 8T bit cells replace the traditional 6T bit cells to store the MSBs because 8T bit cells are more robust at scaled voltages with decoupled read and write paths.

In chapter 4, a systematic data-mining framework that enables a comprehensive understanding of synaptic data characteristics is used to develop a low-power synaptic memory for deep learning. Using an offline data-mining assisted synaptic relationships study, discovered synaptic characteristics including contribution, switching, and association/correlation were used to optimize the memory design. A novel data-driven memory design technique is proposed that can store synaptic weights efficiently and with the ability for self-recovery under memory failures. A 64kbit SRAM is designed that enables considerable power savings of up to 83.2% and with negligible area overhead (3.17%) with minimal loss to classification accuracy (0.72%).

With deep learning becoming popular for developments in artificial intelligence in modern applications such as facial identification, automatic translation, computer vision, self-driving cars, healthcare and education, the need for data privacy is becoming increasingly important. For example, collected health care data can be used in deep learning models to provide personalized methods of prevention, treatment and care, ultimately aiding people who are aging or those with disabilities to address health issues. These learning enabled benefits do

however come at a cost, which partially involves the issue of a serious privacy concern. Sharing personal data carries inherent risk to the individual.

To protect the privacy of an individual's data that are used to train these deep learning models, one technique that has recently been proposed is differentially private deep learning algorithms. This approach adds random noise to the computation while learning so that the output of the model does not significantly depend on any particular training sample. In addition to privacy, as deep learning models grow, the energy and resources needed during the inference process, particularly the memories, have become a major constraint to resource-limited IoT devices. Existing low-power memory design techniques, such as voltage scaling, usually come with large memory failures under low power conditions, which further reduce the accuracy of deep learning systems. Therefore, new memory hardware techniques are needed to consider the trade-off between power efficiency, accuracy, and privacy in order to meet the increasing demands of the edge inference storage in these devices.

In chapter 5, a memory-based technique to enable a suitable accuracy/privacy trade-off that will meet the requirement of different AI applications is presented. The proposed technique will allow for low power operation to enable power savings, thereby enabling an energy-efficient edge inference on edge devices such as IoT sensors.

### **Design Trade-offs and Evaluation**

In order to evaluate how well each of the designed memories perform, simulations were performed to calculate values for power, quality, and area overhead. The relationship between the main three hardware design metrics is shown in Figure 1. By constraining two of the three metrics and improving the third, while also comparing against recent research, it is possible to clearly show improvements of each work against the state of the art. The following chapters

describe the design process for four unique works, each using novel memory optimization techniques, including comparison against the state of the art.

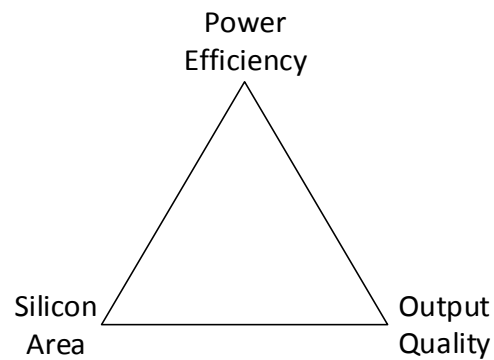


Figure 1. Trade-off between hardware evaluation metrics

All information, tables, and figures in chapters 2, 3, 4, and 5 are either directly taken or adapted, with permission to re-use, from [32], [33], [34], and [35], respectively. The final chapter discusses the comparison of these techniques with state of the art designs in other recent works.



## CHAPTER 2. DATA-PATTERN ENABLED SELF-RECOVERY LOW-POWER STORAGE SYSTEM FOR BIG VIDEO DATA<sup>1</sup>

The growing popularity of powerful mobile devices such as smart phones and tablet devices has resulted in the exponential growth of demand for video applications. However, due to the large video data size and intensive computation, mobile video applications require frequent embedded memory access, which consumes a large amount of power and limits battery life. In this chapter, a low-cost self-recovery video storage system is presented by investigating meaningful data patterns hidden in big video data through introducing data mining techniques to the hardware design process. A two-dimensional data-pattern approach is proposed in order to explore horizontal data-association and vertical data-correlation characteristics. Such data relationship discovery and pattern identification enable a new dimension for the hardware design space and bring self-recovery ability to memories in the presence of bit cell failures. Based on the identified optimal data patterns, a low-cost and efficient SRAM design to enable data self-recovery at low voltages is presented. A 45nm 32kb SRAM is implemented that delivers good video quality at near-threshold voltage (0.5 V) with negligible area overhead (7.94%).

### Embedded Memory Failure Analysis at Near-Threshold Voltage

It has been shown that the computing efficiency is maximized when a circuit is operating at near-threshold voltage [13]. However, at 0.5V (the target near-threshold voltage for this design), SRAM failures become more severe with the increasing process variation. In particular, the RDF effect leads to threshold voltage ( $V_{th}$ ) variation and SRAM cell failures [36]. For the current manufacturing technologies, the failure probability of an SRAM cell ( $P_{fail}$ ) typically

---

<sup>1</sup> The material in this chapter was co-authored by Jonathon Edstrom, Dongliang Chen, and Yifu Gong. Jonathon Edstrom was in charge of all pattern discovery, data analysis, and video quality metric and simulation results. Dongliang Chen and Yifu Gong, provided the presented SRAM hardware design with power simulation results based on the discovered patterns.

ranges between  $10^{-3}$  and  $10^{-2}$ , depending on the bit cell area [13, 37]. The minimum-sized SRAM has highest failure rate of  $10^{-2}$  and larger bit cells have a lower failure probability. With 58% area overhead, the failure rate can be reduced from  $10^{-2}$  to  $10^{-3}$  [37]. Both  $10^{-2}$  (minimum-sized SRAM) and  $10^{-3}$  (upsized SRAM) conditions are considered in the design analysis. It should be noted that, the failure rate can be further optimized using a recently developed priority based sizing technique [38].

To further study the SRAM failure characteristics at low voltage, errors maps for 512 word  $\times$  64 bit SRAM were investigated for  $P_{fail}$  equal to  $10^{-2}$  and  $10^{-3}$ . During the fault injection process, the bits that failed were assumed to be located across the memory cells based on the failure probabilities according to a uniform distribution, introducing embedded memory failures to the decoding process. Using a uniform distribution for simulating the errors is confirmed by memory failure measurements in [39]. The results are shown in Figure 2. SRAM faults are uniformly distributed in the array.

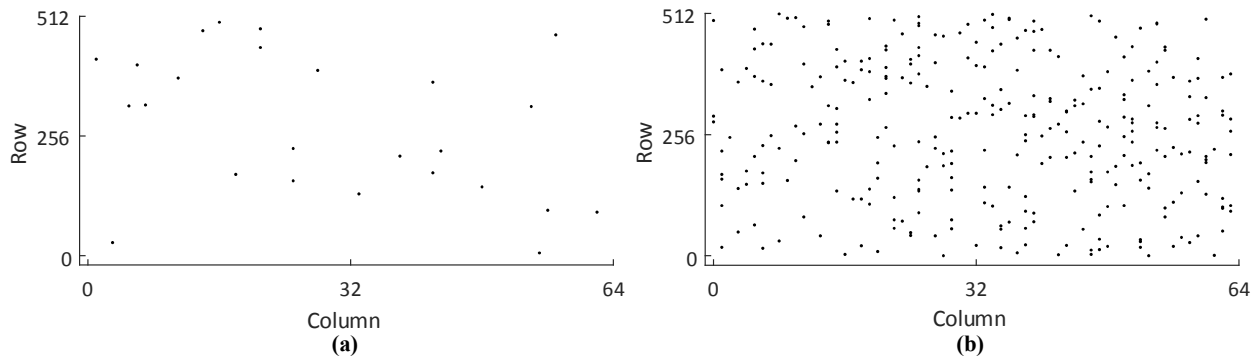


Figure 2. Error maps in SRAM array at 0.5V. (a) Error rate:  $10^{-3}$  and (b) error rate:  $10^{-2}$

The probability of different faults in the same word line (32-bit word) were also analyzed and the results are listed in Table 1. It can be seen that a word line has a low number of faulty cells. The probability of two faults existing in the same word line is only 3.6% when the SRAM

bit cell failure rate is  $10^{-2}$ . Accordingly, in the presence of a memory fault, SRAM may achieve self-recovery based on other bits in the same word line if meaningful bit-level data patterns exist.

Table 1. Fault probability in a 32-bit SRAM word ( $10^9$  Monte Carlo simulations)

Number of faults per word-line	SRAM failure rate: $10^{-3}$ (0.001)	SRAM failure rate: $10^{-2}$ (0.01)
0	96.8523477%	72.7279953%
1	3.0992274%	23.2812509%
2	0.0479198%	3.6012385%
3	0.0005023%	0.3611914%
4	0.0000028%	0.0267011%
5	0%	0.0015432%
6	0%	0.0000756%
7	0%	0.000004%

### Data Pattern Investigation for Self-Recovery

This section presents the data-mining methodology to discover data patterns hidden in video data to enable reliable self-recovery. Specifically, a new two-dimensional (2D) data pattern approach is proposed to explore horizontal data-association and vertical data-correlation characteristics, thereby achieving optimal data patterns.

#### Rule Mining Enabled Horizontal Association

Today's mobile video frames are typically stored and processed in YUV format. The YUV format includes one luma (Y) component, which contains the brightness information of the image, and two chroma components, which contain the blue-difference (Cb) and red-difference (Cr) color information. Figure 3 shows a typical frame of video data stored in embedded memory using a  $352 \times 288$  resolution YUV 4:2:0 video as an example. As shown, each pixel has 8-bit luma data and 8-bit subsampled chroma data. Since video data is stored in on-chip memory as

binary bits, using an association data mining technique to identify horizontal bit-level data patterns is possible.

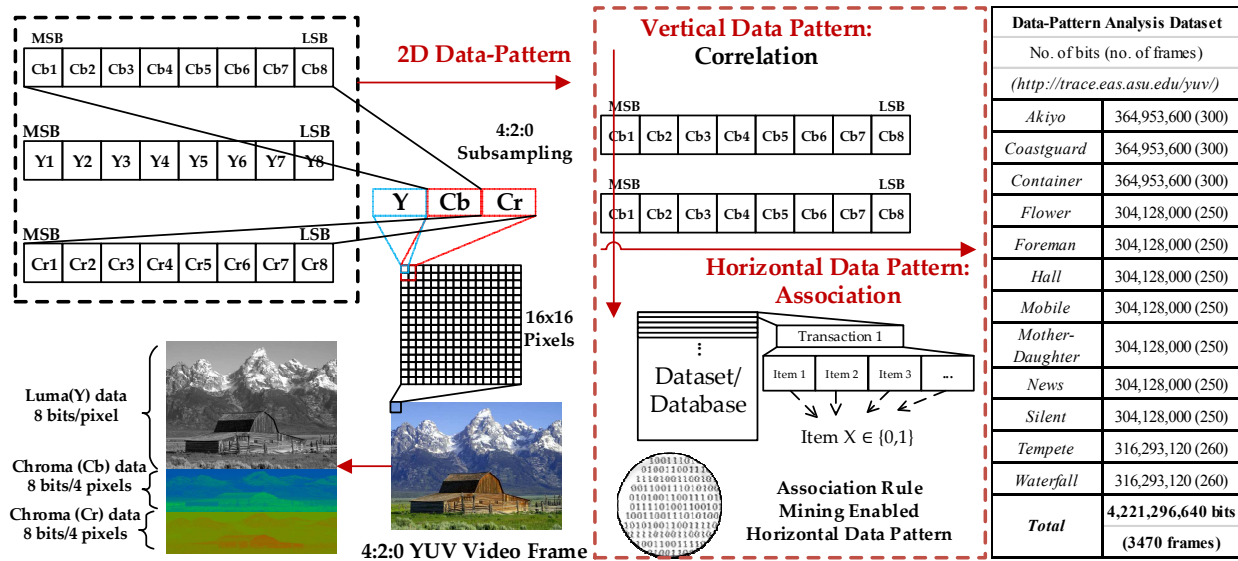


Figure 3. 2D data-pattern enabled self-correction and data pattern analysis dataset

Association rule mining was introduced in 1993 to discover relationships between different variables, called items, in a dataset or database [40]. A complete dataset is made up of many transactions where each transaction contains a set of items. Each item can be associated with a binary attribute, 0 or 1, that is used to distinguish if that item is present or not in its corresponding transaction. This type of data organization is illustrated in Figure 3. Each resulting rule, generated from the association rule mining process, is an implication of the form  $X \rightarrow Y$ , where  $X$  and  $Y$  are disjoint sets of, or individual items. Each rule is also accompanied by collected statistics from the dataset called support and confidence values. The support value for a set of items is the proportion of transactions in the dataset that contains such set of items. The confidence value for an association rule  $X \rightarrow Y$ , is the proportion of transactions that contain  $X$  which also contain  $Y$ , or the conditional probability  $P(Y | X)$ .

To enable association data mining, 12 different video benchmarks were used to build a dataset [41, 42]. In total, the video data size used was 4,221,296,640 bits from 3470 video frames, as shown in Figure 3. Each video data bit was defined as an individual item and Weka [43] was used to perform the well-known Apriori association rule mining algorithm on the large video dataset. Table 2 lists the horizontal data patterns that were discovered for both luma and chroma data based on the video benchmarks.

Table 2. Discovered association rules. Bit 1 (i.e. Y1, Cb1, Cr1) is the MSB

From 12 video benchmarks [41, 42]				From 10,000 Youtube-8M videos [44]			
Association Rules	Confidence	Support	Confidence × Support	Association Rules	Confidence	Support	Confidence × Support
Y2=1 → Y1=0	74.16%	57.15%	42.38%	Y2=1 → Y1=0	76.92%	50.57%	38.90%
Y3=1 → Y1=0	74.28%	52.23%	38.04%	Y3=1 → Y1=0	72.17%	51.52%	37.18%
Y2=0 → Y1=1	22.65%	42.85%	9.71%	Y2=0 → Y1=1	41.51%	71.18%	29.55%
Y1=1 → Y2=0	39.85%	24.22%	9.65%	Y1=1 → Y2=0	71.18%	41.51%	29.55%
Y1=1 → Y3=0	42.86%	24.22%	10.38%	Y1=1 → Y3=0	61.82%	37.01%	22.88%
Cb2=0 → Cb1=1	94.75%	23.23%	22.01%	Cb2=0 → Cb1=1	98.56%	46.53%	45.86%
Cb2=1 → Cb1=0	97.56%	73.64%	71.84%	Cb2=1 → Cb1=0	98.20%	52.65%	51.71%
Cb1=0 → Cb2=1	98.28%	73.64%	72.38%	Cb1=0 → Cb2=1	99.26%	52.69%	52.30%
Cb1=1 → Cb2=0	92.65%	23.23%	21.52%	Cb1=1 → Cb2=0	99.42%	46.56%	46.29%
Cr2=1 → Cr1=0	97.51%	23.52%	22.93%	Cr2=1 → Cr1=0	95.75%	33.66%	32.23%
Cr2=0 → Cr1=1	100.00%	75.88%	75.88%	Cr2=0 → Cr1=1	99.34%	65.38%	64.95%
Cr1=1 → Cr2=0	99.22%	75.88%	75.29%	Cr1=1 → Cr2=0	99.01%	65.43%	64.78%
Cr1=0 → Cr2=1	99.99%	23.52%	23.51%	Cr1=0 → Cr2=1	99.44%	33.71%	33.52%
Cr1=1 → Cr3=0	97.80%	74.80%	73.16%	Cr1=1 → Cr3=0	95.21%	62.61%	59.61%
Cr1=0 → Cr3=1	92.23%	21.69%	20.01%	Cr1=0 → Cr3=1	97.04%	32.62%	31.65%

The video data used for determining optimal rules was further expanded to larger-scale and real video datasets beyond typical benchmarks in order to emulate the use of mobile devices in the environment of big data. Google's YouTube-8M dataset [44], one of the largest video databases to date, was used for this purpose. Specifically, 10,000 unique videos from the YouTube-8M dataset, with a total data size of 57.6 gigabytes, representing 500,000 individual frames, was analyzed using data mining methods. A script was written that would download the 10,000 videos from the approximately 7 million available URLs provided in the YouTube-8M dataset. After each video was downloaded, 50 contiguous frames were randomly selected from the video and were converted from the .mp4 file format to the raw .yuv format using the FFmpeg decoder [45] for data mining analysis. In order to support large-scale video data processing, these data mining calculations were performed on the Thunder cluster at the Center for Computationally Assisted Science and Technology (CCAST) at North Dakota State University, which consists of approximately 100 compute nodes with a theoretical peak performance of around 150 teraflops. As illustrated in Figure 3, each video data bit is defined as an individual item and the well-known Apriori algorithm was used on the video datasets to gather both confidence and support metric calculations. The average results obtained for the horizontal data patterns are also listed in Table 2. It can be seen that the association rules obtained from the video benchmarks are very general and also exist within the large-scale video dataset.

### **Vertical Correlation**

Vertical data correlation characteristics of multimedia applications have been studied by many researchers [25, 46]. These works have shown that the MSBs of the video data have a strong correlation with neighboring pixels, and their switching probabilities are very low. Based on the video benchmarks, Table 3 lists the correlation probability of the MSBs (Y1, Cb1, Cr1) in

neighboring pixels is over 93%, while the LSBs have around 53% correlation probability in the worst case (i.e. Cb8). A similar trend in probabilities can be seen for the YouTube-8M videos. The MSBs in neighboring pixels have a very strong correlation (with probability over 90%), but LSBs display more of a random behavior and have little correlation with neighboring pixels.

Table 3. Vertical correlation probabilities

Correlation probabilities from 12 video benchmarks					
Y1	96.72%	Cb1	93.79%	Cr1	93.78%
Y2	93.22%	Cb2	92.87%	Cr2	93.58%
Y3	87.79%	Cb3	90.77%	Cr3	92.34%
Y4	80.86%	Cb4	85.45%	Cr4	88.35%
Y5	73.97%	Cb5	77.95%	Cr5	81.56%
Y6	67.42%	Cb6	69.30%	Cr6	73.18%
Y7	61.78%	Cb7	59.99%	Cr7	63.50%
Y8	58.52%	Cb8	53.25%	Cr8	55.16%
Correlation probabilities from 10,000 YouTube-8M videos					
Y1	91.55%	Cb1	90.10%	Cr1	90.41%
Y2	84.64%	Cb2	89.87%	Cr2	90.12%
Y3	77.23%	Cb3	88.67%	Cr3	88.75%
Y4	68.87%	Cb4	84.86%	Cr4	85.11%
Y5	60.05%	Cb5	78.11%	Cr5	78.72%
Y6	51.49%	Cb6	68.87%	Cr6	70.11%
Y7	44.14%	Cb7	58.81%	Cr7	60.56%
Y8	38.78%	Cb8	50.84%	Cr8	52.62%

Power saving techniques involving the correlation have been used in previous works for bit prediction where no transistor switching results in power savings [9] and attempting to load the same value (i.e. reading continuous 0s or 1s) from memory bit cells in order to eliminate the cost of pre-charging if the correct value is read out from the previous bit-line read [25]. This work uses the correlation property of YUV data through the use of a novel bit correction technique that attempts to correct memory faults with high precision. By comparing the

correlation percentages and the association rules that have been identified, the best combination of association rules and correlation between bits can be constructed for an optimal pattern for data self-recovery.

### Optimal Data Patterns for Self-Recovery

In order to select an optimal data pattern from association and correlation, the *Weighted Confidence*, based on the support and confidence of a particular rule, is defined as follows:

$$\begin{aligned} \text{Weighted Confidence} &= \text{Confidence}(\text{Rule}) \times \text{Support}(\text{Rule}) + \\ &\text{Confidence}(\text{Complement Rule}) \times \text{Support}(\text{Complement Rule}) \end{aligned} \quad (1)$$

For example, the *Weighted Confidence* of the association rule  $\overline{Cr1} \rightarrow Cr2$  can be expressed as:

$$\begin{aligned} \text{Weighted Confidence of } (\overline{Cr1} \rightarrow Cr2) & \\ &= \text{Confidence}(\overline{Cr1} = 0 \rightarrow Cr2 = 1) \times \text{Support}(\overline{Cr1} = 0 \rightarrow Cr2 = 1) \\ &+ \text{Confidence}(Cr1 = 1 \rightarrow Cr2 = 0) \times \text{Support}(Cr1 = 1 \rightarrow Cr2 = 0) \\ &= 0.9999 \times 0.2352 + 0.9922 \times 0.7588 = 0.9880 \end{aligned} \quad (2)$$

This parameter is then used to compare to the sum of the correlation values for 0 and 1 non-switching, which is equal to the correlation. This is equivalent to the *Weight Confidence* calculation, but instead uses the individual bit value (0 or 1) correlation percentages and is calculated as follows:

$$\begin{aligned} \text{Correlation} &= \text{Confidence}(\text{Bit}_{\text{previous}} = 0 \rightarrow \text{Bit}_{\text{current}} = 0) + \\ &\text{Confidence}(\text{Bit}_{\text{previous}} = 1 \rightarrow \text{Bit}_{\text{current}} = 1) \end{aligned} \quad (3)$$

where  $\text{Bit}_{\text{previous}}$  and  $\text{Bit}_{\text{current}}$  represent the video data bits in the same position of two neighboring pixels.

As an example, the correlation of Cr2 can be calculated as follows:



$$\begin{aligned}
& \text{Correlation of } Cr2 = \text{Confidence}(Cr2_{previous} = 0 \rightarrow Cr2_{current} = 0) + \\
& \qquad \qquad \qquad \text{Confidence}(Cr2_{previous} = 1 \rightarrow Cr2_{current} = 1) \\
& = 0.2091 + 0.726 = 0.935 \tag{4}
\end{aligned}$$

Accordingly, the optimal bit-level data patterns with high prediction rate to enable self-recovery were calculated. 25 videos from the YouTube-8M dataset, separate from the 10,000 videos used in the training dataset, were used to verify the correction prediction percentage shown in Table 4. These videos are obtained using the same method as previously used, but are unique from the previous 10,000 videos to ensure the rules that are employed work properly for correction. An analysis of the different portions of the image show that luma data is more random and has less association with other bits in the same pixel, and the optimal data patterns are all from correlation.

Table 4. Optimal data patterns from 25 YouTube-8M videos

Y bits	Optimal Data Patterns	Correct Prediction	Cb bits	Optimal Data Patterns	Correct Prediction	Cr bits	Optimal Data Patterns	Correct Prediction
Y1	Correlation (Y1 <sub>previous</sub> )	91.53%	Cb1	Association ( $\overline{Cb2} \rightarrow Cb1$ )	98.60%	Cr1	Association ( $\overline{Cr2} \rightarrow Cr1$ )	96.72%
Y2	Correlation (Y2 <sub>previous</sub> )	82.67%	Cb2	Association ( $\overline{Cb1} \rightarrow Cb2$ )	99.79%	Cr2	Association ( $\overline{Cr1} \rightarrow Cr2$ )	97.77%
Y3	Correlation (Y3 <sub>previous</sub> )	76.27%	Cb3	Correlation (Cb3 <sub>previous</sub> )	88.46%	Cr3	Association ( $\overline{Cr1} \rightarrow Cr3$ )	93.86%
Y4	Correlation (Y4 <sub>previous</sub> )	67.64%	Cb4	Correlation (Cb4 <sub>previous</sub> )	84.31%	Cr4	Correlation (Cr4 <sub>previous</sub> )	83.64%
Y5	Correlation (Y5 <sub>previous</sub> )	59.24%	Cb5	Correlation (Cb5 <sub>previous</sub> )	78.53%	Cr5	Correlation (Cr5 <sub>previous</sub> )	78.35%
Y6	Correlation (Y6 <sub>previous</sub> )	51.75%	Cb6	Correlation (Cb6 <sub>previous</sub> )	69.40%	Cr6	Correlation (Cr6 <sub>previous</sub> )	68.80%
Y7	Correlation (Y7 <sub>previous</sub> )	44.47%	Cb7	Correlation (Cb7 <sub>previous</sub> )	59.40%	Cr7	Correlation (Cr7 <sub>previous</sub> )	59.73%
Y8	Correlation (Y8 <sub>previous</sub> )	38.41%	Cb8	Correlation (Cb8 <sub>previous</sub> )	51.13%	Cr8	Correlation (Cr8 <sub>previous</sub> )	52.96%

## Recovery Failure Caused by Double Faults in Data Patterns

Since the discovered optimal data patterns used for self-recovery exist between two bits in the same word line, it may cause recovery failures if both of the two bits in a pattern fail simultaneously. Table 5 lists the recovery failure rate. It shows that DPSR has good reliability with extremely low self-recovery rates (less than 0.2%). This is due to the fact that there is low probability of having multiple faults in the same word line, as discussed earlier.

Table 5. DPSR recovery failure rates

Double Word Line Faults	SRAM $P_{\text{fail}}: 10^{-3}$ (0.001)	SRAM $P_{\text{fail}}: 10^{-2}$ (0.01)
Correlation Faults	0.0010899%	0.1077362%
Association Faults	0.0005957%	0.0587964%
DPSR Recovery Failure	0.0016856%	0.1665326%

## DPSR Hardware Implementation

Utilizing the obtained optimal bit-level data patterns, a simple but efficient DPSR hardware design with low implementation cost is presented. Figure 4 shows the array architecture of the proposed DPSR SRAM, where the total array size is 32 kbits and there are four blocks with  $256 \text{ words} \times 32 \text{ bits}$ . In the design, both luma data and chroma data will be stored in the same SRAM but in different blocks. Block 1 and block 2 will be used to store the luma data and each word line will store the luma data of 4 pixels. Block 3 and block 4 will be used to store the chroma data and each word line will store the chroma data of 2 pixels.

Regarding the luma data stored in blocks 1 and 2, based on the optimal luma patterns obtained previously, vertical correlation rules (i.e. luma data of the previous pixel) will be used for recovering the data of the current pixel. Since SRAM read operations are row-wise, reading two physical rows will cause a considerable performance penalty. Accordingly, the vertical

correlation based luma self-recovery is adapted to a hardware-friendly design scheme. Each word line stores the luma data of 4 pixels and uses its neighboring pixel in the same row for data correction in the current pixel. As an example, if a data bit in pixel 1 has a failure (see Figure 4), the corresponding bit in pixel 2 is used for recovery; if a bit in pixel 4 has a failure, the corresponding bit in pixel 3 is used for recovery (i.e the neighboring pixel in the same row).

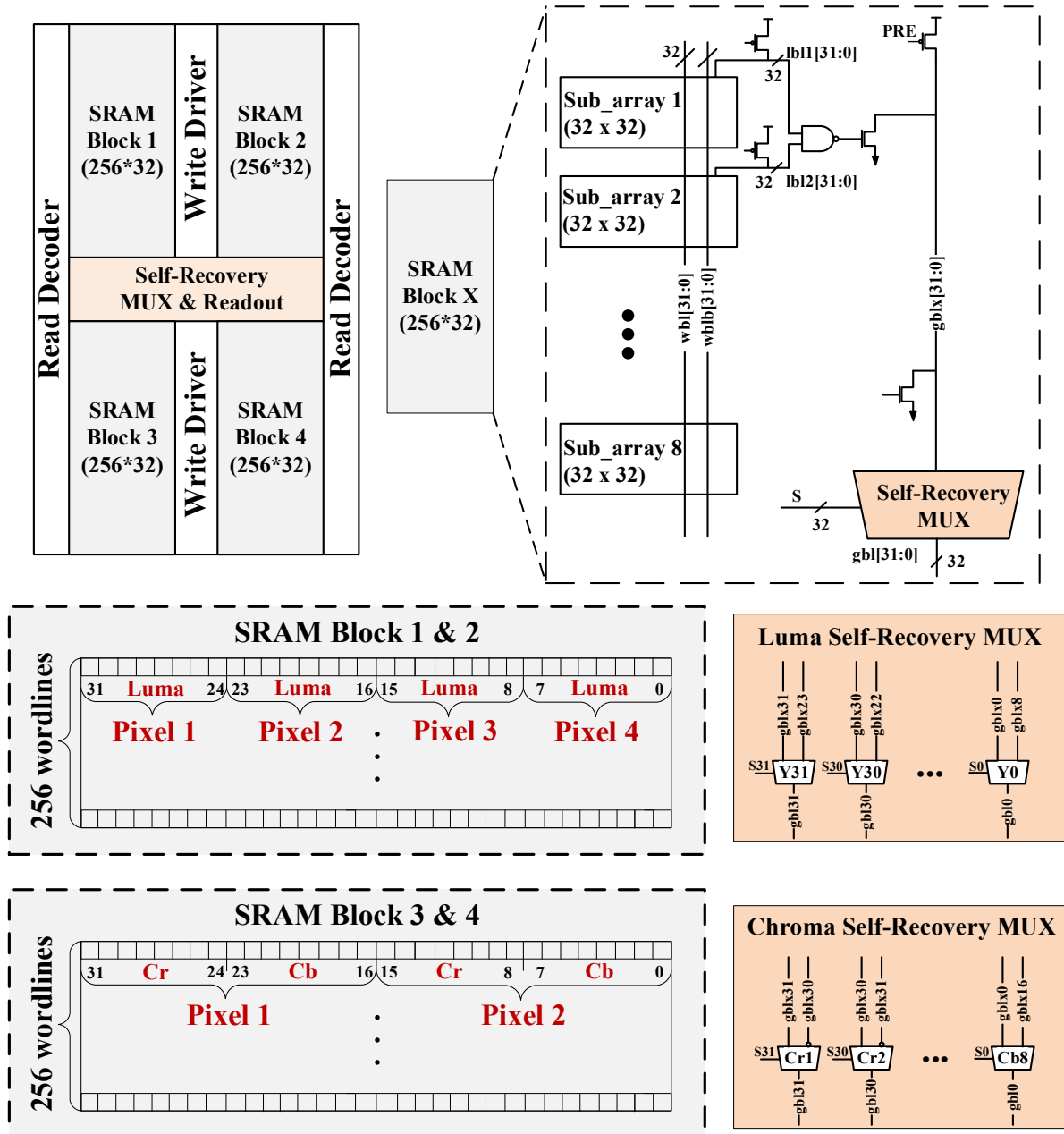


Figure 4. Proposed DPSR with data self-recovery ability

To verify the DPSR design will maximize the correct bit predictions, calculations were performed for the correct prediction percentage for predicting each bit using both the previous and next pixel's corresponding bit. These calculations showed that they were approximately equal values, with both having ~79.4% average correct prediction for all luma bits based on calculations from all samples in the training and verification testing video benchmarks [41, 42].

Chroma data self-recovery is implemented in SRAM block 3 and block 4 using the optimal chroma patterns. As shown in Figure 4, each word line stores two pixels of chroma data with both Cr and Cb. Both vertical correlation rules and horizontal data pattern rules are used for the chroma data self-recovery (see Table 4). Similarly, for vertical correction based recovery rules, the neighboring pixel stored in the same row for data corrections are used to avoid a penalty to performance. For example, if Cb1 in pixel 1 has a failure (see Figure 4), the inverted value of Cb2 in the same pixel will be used for recovery. If Cr4 in pixel 1 fails, Cr4 in pixel 2, which is stored in the same row, will be used for recovery.

As shown in Figure 4, a hierarchical RBL scheme (i.e. local RBL and global RBL) is applied to reduce the access time of the memory. The self-recovery logic of DPSR can be simply implemented by connecting MUXs to RBLs of the conventional SRAM design. Each global bit-line (i.e. gbl in Figure 4) is connected to a MUX which is controlled by the received fault positions. If a fault is indicated, self-recovery is enabled by selecting the data pattern. The fault position information is used as the select signal of the MUX in order to control which bit value will be output. Similar to other existing fault position aware mitigation techniques, DPSR receives pre-determined locations of the faulty bits in the SRAM array, which is usually executed during post fabrication testing or Power-On Self-Test (POST) [13, 47, 48]. This testing process can also be used to track temporal degradation caused by memory failures such as the

aging effect. The evaluation results in the following sections show that the DPSR SRAM design also achieves smaller silicon area overhead, while delivering good output quality at near threshold voltage.

### Evaluation Methodology and Results

To evaluate the effectiveness of the proposed technique, a 32kb SRAM was implemented using a high-performance 45nm FreePDK CMOS process [49] to meet the multi-megahertz performance requirement of today’s mobile video decoders.

#### Performance

The performance of the proposed DPSR design was first evaluated. Due to the added MUXs, the read access time of DPSR increases from 0.27ns to 0.31ns, which is fast enough to deliver high-quality video formats including 4K and 8K ultra high-definition applications [50].

#### Layout

As discussed before, embedded SRAMs typically occupy a large portion of silicon area within a video chip, and therefore the cost of the embedded SRAM is an important design concern. Figure 5 shows the layout of DPSR. Each added self-recovery logic MUX occupies an area of  $18.79\mu\text{m} \times 43.47\mu\text{m}$ , resulting in 7.94% area overhead. It should also be noted that, the self-recovery logic is added to the RBLs and increasing the number of words in a memory is beneficial in reducing the overall area overhead.

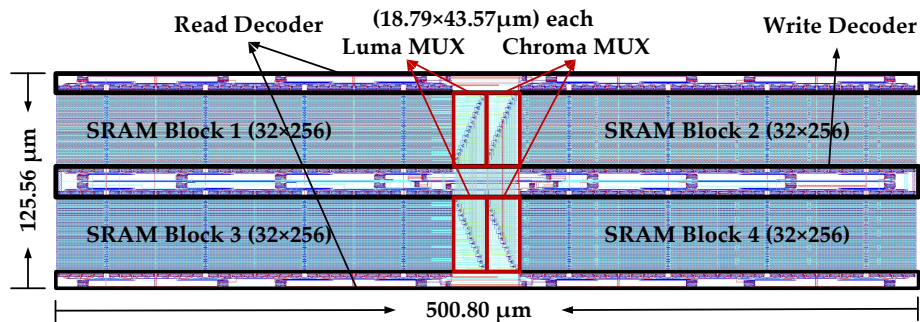


Figure 5. DPSR layout design

## Power Efficiency

In order to evaluate the power efficiency of the proposed technique, the power consumption of the memory is modeled as follows:

$$P_{Dynamic} = \frac{\sum_{j=0}^{31} \sum_{i=0,1} [P_j(i) \cdot (R(i) + W(i))]}{2} \quad (5)$$

$$P_{Leak} = \sum_{d=0}^{1023} \sum_{j=0}^{31} \sum_{i=0,1} L_j(i) \quad (6)$$

where  $P_{Dynamic}$  and  $P_{Leak}$  are the dynamic and leakage power consumption, respectively.  $i$  is the value stored in SRAM,  $j$  is the bit number in a word, which is from 0 to 31.  $P_j(i)$  is the probability of a data bit  $j$  to be 0 or 1, which is extracted from various video benchmarks.  $R(i)$ ,  $W(i)$ , and  $L(i)$  are the read power, write power, and leakage power consumption for a single SRAM bit cell, respectively, that stores a particular data bit value  $i$ . Figure 6 compares the power consumption in different memory operations. As expected, all power components decrease as the voltage scales from 1.0V to 0.5V. It should be noted that the power consumption overhead caused by the self-correction logic in the proposed technique is negligible as compared to the power reduction enabled by reducing voltage to near-threshold voltage, since the dynamic and leakage power consumption scale at a quadratic and linear rate with the voltage, respectively.

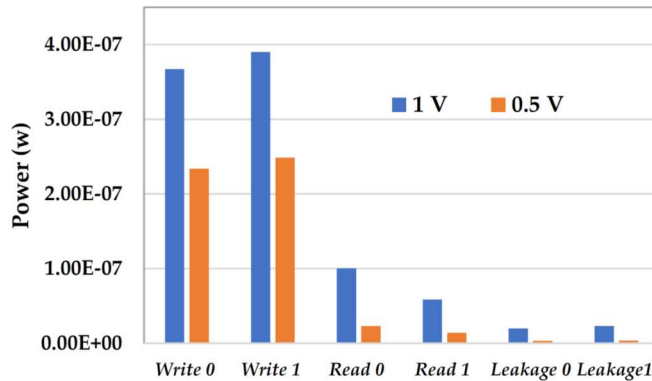


Figure 6. Power consumption of different memory operations

The proposed memory at 0.5V consumes 219 $\mu$ W dynamic power and 193 $\mu$ W leakage power. As compared to the conventional memory design at 1.0V, the proposed design has 81.52% dynamic power savings and 82.45% leakage power savings.

### Video Output Quality Analysis

Different from the video benchmark datasets used previously for this design, a new video benchmark dataset is organized for verification: 3 videos from [41] and 5 videos from [42]. To evaluate the video quality, the well-known PSNR metric is adopted, which is defined as [25]:

$$PSNR = 10 \cdot \log_{10} \left( \frac{255^2}{MSE} \right) \quad (7)$$

where MSE is the mean squared error between the original videos (*Org*) and the degraded videos (*Deg*), expressed as:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [Org(i, j) - Deg(i, j)]^2 \quad (8)$$

Researchers have shown that a PSNR of 30dB or higher for a video are considered to be acceptable [13]. Table 6 compares the PSNR values using different techniques with a  $P_{fail}$  (i.e. the failure probability) of  $10^{-2}$  for minimum-sized SRAM designs and  $10^{-3}$  for upsized SRAM with 58% area overhead [37]. In addition to video benchmarks, 10 YouTube-8M videos from the 25 videos used for verification earlier in Table 4 (separate from the 10,000 videos used in the training dataset) are used for calculating the video metrics presented. Due to the limited space, Figure 7 shows six video output images with a memory failure rate of  $10^{-2}$  when failures are injected. It can be seen that DPSR has good recovery precision and can deliver good video quality with a PSNR over 35dB, even for minimum sized SRAM. Accordingly, DPSR achieves good video output quality at near-threshold voltages.



Video	Original Video	Conventional ( $P_{fail} = 0.01$ )	DPSR ( $P_{fail} = 0.01$ )	Shift ( $P_{fail} = 0.01$ ) [23]
city	 <i>PSNR: 36.8039 SSIM: 0.9306</i>	 <i>PSNR: 24.5045 SSIM: 0.5739</i>	 <i>PSNR: 33.7729 SSIM: 0.9095</i>	 <i>PSNR: 36.7801 SSIM: 0.9290</i>
crew	 <i>PSNR: 37.1454 SSIM: 0.9078</i>	 <i>PSNR: 24.5212 SSIM: 0.5142</i>	 <i>PSNR: 35.5632 SSIM: 0.8901</i>	 <i>PSNR: 37.1197 SSIM: 0.9060</i>
football	 <i>PSNR: 36.5037 SSIM: 0.9163</i>	 <i>PSNR: 24.4878 SSIM: 0.5542</i>	 <i>PSNR: 34.6731 SSIM: 0.9046</i>	 <i>PSNR: 36.4816 SSIM: 0.9148</i>
Concert	 -	 <i>PSNR: 24.7459 SSIM: 0.6161</i>	 <i>PSNR: 39.8358 SSIM: 0.9887</i>	 <i>PSNR: 59.4008 SSIM: 0.9988</i>
Game	 -	 <i>PSNR: 24.7593 SSIM: 0.5834</i>	 <i>PSNR: 39.6992 SSIM: 0.9839</i>	 <i>PSNR: 59.4008 SSIM: 0.9987</i>
Electric Guitar	 -	 <i>PSNR: 24.7528 SSIM: 0.5580</i>	 <i>PSNR: 42.5844 SSIM: 0.9884</i>	 <i>PSNR: 59.4008 SSIM: 0.9985</i>

Figure 7. Video output using different video storage techniques



The PSNR metric has been used extensively to describe video output quality in a quantitative way, but recent efforts to capture the true human perception show that it may not accurately describe the actual video quality a human perceives [51]. This is due to the fact that PSNR is based on the summation of error for every pixel's luminance and chrominance component values, and this alone is not necessarily a good estimate to the user's perception of the video. Analyzing the video quality using SSIM is a method that is more aware of the user's perception since it includes calculations for luminance, contrast, and structural changes in the video. The general form of the SSIM equation is defined as [51]:

$$SSIM(x, y) = [l(x, y)]^\alpha \cdot [c(x, y)]^\beta \cdot [s(x, y)]^\gamma = \frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)} \quad (9)$$

where  $l(x,y)$ , the luminance comparison, is a function of the mean intensities,  $\mu_x$  and  $\mu_y$ ,  $c(x,y)$ , the contrast comparison is a function of the standard deviations,  $\sigma_x$  and  $\sigma_y$ , and  $s(x,y)$ , the structural comparison, is a function of the correlation between  $x$  and  $y$ , or  $\sigma_{xy}$ . Setting the values of  $\alpha = \beta = \gamma = 1$  in the original equation results in the second equation.  $C_1$  ( $C_2$ ) is a constant that is included to avoid instabilities when the sum of the means (standard deviations) squared is equal to the values near zero. The value of the SSIM is in the range 0 to 1. As the value of  $SSIM(x,y)$  gets closer to 1, the quality of the video  $y$  more closely matches the quality of video  $x$ . For our testing purposes, video  $x$  is the raw, uncompressed YUV video, before the decoding process, and video  $y$  is the post decoded YUV video that may or may not have other bit shifting or correction changes performed on it.

The results of these SSIM calculations for conventional and DPSR are listed in Table 7. The video output quality of the proposed DPSR method has a significant increase in SSIM over the no failure, conventional memory at scaled voltages.

Table 6. Video PSNR metric comparison

Dataset	Videos	conventional ( $P_{fail} = 0.001$ )	DPSR ( $P_{fail} = 0.001$ )	conventional ( $P_{fail} = 0.01$ )	DPSR ( $P_{fail} = 0.01$ )	Ref. [23] ( $P_{fail} = 0.001$ )	Ref. [23] ( $P_{fail} = 0.01$ )
<b>Video Benchmarks</b>	<i>akiyo</i>	33.762219	<b>40.641272</b>	24.676287	<b>36.639433</b>	41.248639	41.185088
	<i>bus</i>	32.102969	<b>35.405863</b>	24.410622	<b>33.373556</b>	35.706569	35.689801
	<i>city</i>	32.550805	<b>36.360825</b>	24.426879	<b>33.772872</b>	36.801408	36.780126
	<i>coastguard</i>	32.090258	<b>35.489524</b>	24.426879	<b>34.265358</b>	35.667736	35.650842
	<i>crew</i>	32.680147	<b>36.928508</b>	24.521212	<b>35.563219</b>	37.142670	37.119667
	<i>football</i>	32.439115	<b>36.255904</b>	24.487795	<b>34.673071</b>	36.501345	36.481558
	<i>foreman</i>	32.71063	<b>36.878115</b>	24.529656	<b>35.022773</b>	37.211848	37.188112
	<i>sign_irene</i>	33.253495	<b>38.980559</b>	24.590776	<b>36.573802</b>	38.976183	38.940649
<b>YouTube 8M Dataset</b>	<i>Running</i>	34.843802	<b>47.751093</b>	27.751663	<b>37.896356</b>	69.178843	59.400849
	<i>Concert</i>	34.843123	<b>50.617823</b>	24.745933	<b>39.835772</b>	69.178843	59.400849
	<i>Music Video</i>	34.842942	<b>48.993861</b>	24.765553	<b>37.908828</b>	69.178843	59.400849
	<i>Festival</i>	34.843240	<b>45.838237</b>	24.892104	<b>35.958557</b>	69.178843	59.400849
	<i>Game</i>	34.843259	<b>49.286247</b>	24.759353	<b>39.699233</b>	69.178843	59.400849
	<i>Electric Guitar</i>	34.843014	<b>51.566521</b>	24.752845	<b>42.584377</b>	69.178843	59.400849
	<i>Snow</i>	34.844445	<b>50.725480</b>	24.761392	<b>40.861991</b>	69.178843	59.400849
	<i>Flute</i>	34.842227	<b>53.769387</b>	24.755972	<b>44.158630</b>	69.178843	59.400849
	<i>Vehicle</i>	34.843032	<b>50.015065</b>	24.741031	<b>42.251862</b>	69.178843	59.400849
<i>Planet</i>	34.843295	<b>53.306924</b>	24.760113	<b>44.022668</b>	69.178843	59.400849	

Table 7. Video SSIM metric comparison

Dataset	Videos	Conventional ( $P_{fail} = 0.001$ )	DPSR ( $P_{fail} = 0.001$ )	conventional ( $P_{fail} = 0.01$ )	DPSR ( $P_{fail} = 0.01$ )	Ref. [13] ( $P_{fail} = 0.001$ )	Ref. [13] ( $P_{fail} = 0.01$ )
<b>Video benchmarks</b>	<i>akiyo</i>	0.895568	<b>0.960037</b>	0.524455	<b>0.943273</b>	0.961509	0.959164
	<i>bus</i>	0.884369	<b>0.928646</b>	0.615363	<b>0.917352</b>	0.929814	0.928514
	<i>city</i>	0.879284	<b>0.928319</b>	0.573905	<b>0.909453</b>	0.93045	0.929045
	<i>coastguard</i>	0.872335	<b>0.919269</b>	0.587307	<b>0.910952</b>	0.920116	0.918561
	<i>crew</i>	0.850047	<b>0.905746</b>	0.514167	<b>0.890076</b>	0.907585	0.906039
	<i>football</i>	0.863992	<b>0.914951</b>	0.554214	<b>0.904554</b>	0.91613	0.914782
	<i>foreman</i>	0.865366	<b>0.920302</b>	0.539163	<b>0.908825</b>	0.921568	0.919849
	<i>sign_irene</i>	0.879546	<b>0.940751</b>	0.521892	<b>0.928188</b>	0.942161	0.940099
<b>YouTube 8M Dataset</b>	<i>Running</i>	0.949418	<b>0.997716</b>	0.631449	<b>0.979334</b>	0.999886	0.998972
	<i>Concert</i>	0.945931	<b>0.998801</b>	0.616055	<b>0.988679</b>	0.999874	0.998849
	<i>Music Video</i>	0.945398	<b>0.998251</b>	0.607637	<b>0.980091</b>	0.999876	0.998857
	<i>Festival</i>	0.953149	<b>0.998222</b>	0.660847	<b>0.983623</b>	0.999892	0.998987
	<i>Game</i>	0.941848	<b>0.998281</b>	0.583433	<b>0.983909</b>	0.999855	0.998658
	<i>Electric Guitar</i>	0.937000	<b>0.998636</b>	0.558013	<b>0.988400</b>	0.999842	0.998543
	<i>Snow</i>	0.939508	<b>0.998665</b>	0.573617	<b>0.987401</b>	0.999850	0.998656
	<i>Flute</i>	0.936771	<b>0.999146</b>	0.551228	<b>0.992497</b>	0.999848	0.998629
	<i>Vehicle</i>	0.942345	<b>0.999002</b>	0.588818	<b>0.992204</b>	0.999855	0.998681
<i>Planet</i>	0.931941	<b>0.998594</b>	0.533084	<b>0.991441</b>	0.999818	0.998322	

## **Comparison with Prior Work**

With data-pattern enabled self-recovery ability, DPSR exhibits low implementation cost (7.94%) and reliable operation at near-threshold voltage to achieve maximum energy efficiency.

### ***Comparing with State-of-the-Art Data-Shifting [23]***

Table 7 also compares the video output quality of the proposed DPSR and the data-shifting technique presented in [23]. As shown, the data-shifting technique [23] has slightly better quality in terms of PSNR and SSIM metrics as compared to the proposed DPSR technique, but is realized with large area overhead (~14%). This large overhead is due to the fact that the shifting scheme needs to calculate the shift values based on the received fault positions and then perform shifting to store LSBs in the identified faulty bit cells.

### ***Comparing with State-of-the-Art Data-Squeezing [13]***

The data squeezing technique presented in [13] is another recently developed memory failure mitigation technique. Based on the observation that, for many general purpose applications, the last-level cache contains large amounts of null data, this technique compresses null subblocks so that they can be allocated to memory entries with faulty cells. This technique works well for register files and caches for general purpose applications, which store as high as 79.23% zeros as discussed in [52]. However, it is not suitable for videos because the 8-bit video pixel data varies a lot between 0 and 255, which is difficult for zero compression.

### ***Comparing with State-of-the-Art Error Correction Code***

ECCs have also been studied in ultra-low voltage contexts to protect against memory failures [53]. For similar redundancy based repair mechanisms to implement ECC, the capacity of a memory needs to be increased or part of its effective capacity has to be sacrificed to store check bits. In addition to memory space overhead, complex logic for ECC encoding and

decoding must be added, which brings significant implementation penalty. For example, by using orthogonal Latin square codes discussed in [53], half of the memory capacity is used to store ECC bits.

### DPSR Concluding Remarks

In this developed big-data enabled memory technique, the general data patterns existing in large scale videos have been identified, which are used to achieve self-correction in the presence of memory failures. The overhead of the developed self-correction logic is significantly reduced as compared to existing techniques. Table 8 displays a comparison of the performance of the proposed DPSR [32] memory design for big video data against other recent state of the art techniques. Using data-pattern enabled self-recovery, DPSR has the lowest implementation cost (3.97% area overhead) and has reliable operation at near-threshold voltage, allowing for maximum energy efficiency. DPSR delivers the best video quality output, except for [23], which is realized with large area overhead (~14%).

Table 8. DPSR comparison with prior works on low power SRAM

	TCASl'12 [16]	DAC'15 [23]	TC'16 [13]	<b>DPSR [32]</b>
fault-position awareness	No	Yes	Yes	<b>Yes</b>
Low-power techniques	bit-cell Sizing	data-shifting	data-squeezing	<b>data-pattern enabled self-recovery</b>
bit-cell modified	Yes	No	No	<b>No</b>
near-threshold operation	No (0.9V)	Yes (-)	Yes (0.5V)	<b>Yes (0.5V)</b>
additional logic needed	No	LUTs and shifter	Rearrangement logic and tag array, comparator, Mux	<b>MUX</b>
performance overhead	-	-	extra clock (for decompression)	<b>0.04 ns</b>
video quality	acceptable	good	-	<b>good</b>
area overhead	11-65%	14%	6.3%	<b>3.97%</b>

## **CHAPTER 3. CONTENT-ADAPTIVE MEMORY FOR VIEWER-AWARE ENERGY-QUALITY SCALABLE MOBILE VIDEO SYSTEMS<sup>2</sup>**

Mobile devices are becoming ever more popular for streaming videos, which account for the majority of all data traffic on the internet. Memory is a critical component in mobile video processing systems, increasingly dominating power consumption. Today, memory designers are still focusing on hardware-level power optimization techniques, which usually come with significant implementation cost (e.g., silicon area overhead or performance penalty). In this chapter, a video content-aware memory technique for power-quality trade-off from viewers' perspectives is proposed. Based on the influence of video macroblock characteristics on viewer experience, two simple and effective models - decision tree and logistic regression – are developed in order to enable hardware adaptation. A novel viewer-aware bit-truncation technique has also been implemented, which minimizes the impact on viewer experience, while introducing energy-quality adaptation to the video storage.

### **Influence of Video Content on Viewer Experience**

#### **Mobile Video Memory System**

Video streaming has become the most important energy-intensive application used in mobile devices [30]. Figure 8 (a) shows the block diagram of a H.264 video decoding and display system [54]. After parsing the compressed bit stream, the inter predictor uses the reconstructed frames stored in the reference frame buffer and the transmitted motion vectors to construct new frames. After the frames are decoded, the display controller sends them from the

---

<sup>2</sup> The material in this chapter was co-authored by Jonathon Edstrom, Yifu Gong, and Ali Haidous. Jonathon Edstrom, was in charge of all data analysis, video quality metric and simulation results. Yifu Gong, provided the presented SRAM hardware design with power simulation results based on the data analysis and software simulations. Ali Haidous, provided information on the memory architecture and influence of truncation within different memories of the video decoder process.

frame buffer to the display panel periodically. During this process, multiple memories are needed for storing the intermediate and final results of the frame data, as listed in Table 9.

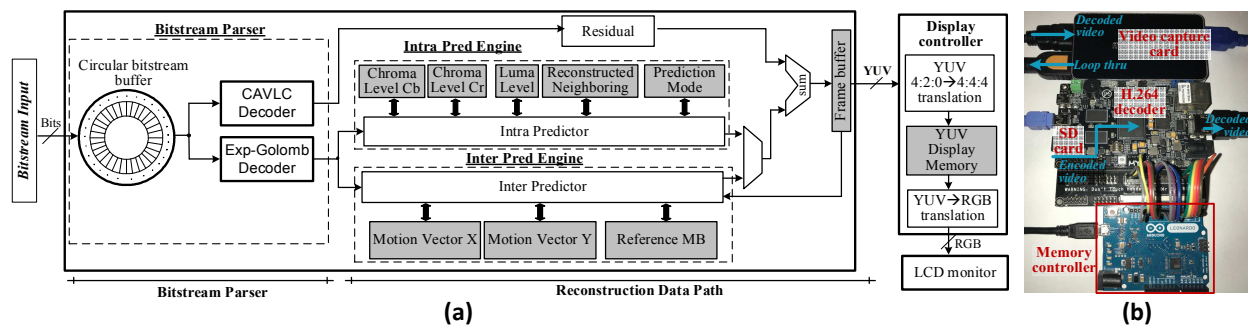


Figure 8. Mobile video memory architecture.

To evaluate the contribution of different memories to the output video quality, a video decoder and display system was developed, as shown in Figure 8 (b). For the memories listed in Table 8, the bit truncation technique from [29] was applied by disabling the LSBs [55, 56] and then the output video was captured for quality evaluation. Specifically, LSB truncation starting with one bit with a maximum of five bits have been applied to each video memory. The encoded bit stream, which resided on an onboard SD card, is decoded using a Xilinx Zynq 7020 FPGA based H.264 decoder. An Arduino-based memory controller is implemented to select the specific memory for truncation as well as the number of truncated LSBs. A video capture card is utilized to capture the video output over the HDMI output for evaluation.

Table 9. Video memories and their functionality

Video Memories	Size in Bits (Width×Depth)	Memory Functionality
Chroma Level Cb	32 x 8	Stores the blue-difference color space bottom line pixels for up macroblocks
Chroma Level Cr	32 x 8	Stores the red-difference color space bottom line pixels for up macroblocks
Luminosity Level	32 x 8	Stores the luminosity color space bottom line pixels for up macroblocks
Reconstructed Neighboring	32 x 7	Stores neighboring pixels of a luma block after the current macroblock is coded and reconstructed
Prediction Mode	16 x 7	Stores the current macroblock prediction mode for 4x4 blocks
Motion Vector X	64 x 7	Stores the horizontal motion vector prediction calculation of surrounding blocks' motion data
Motion Vector Y	64 x 7	Stores the vertical motion vector prediction calculation of surrounding blocks' motion data
Reference Macroblock	8 x 8	Stores the reference I, SI, P, or SP macroblock used for inter prediction
<b>Frame buffer</b>	<b>64 x 512</b>	<b>Stores the current and previous decoded frames for prediction and display, respectively</b>
Y Display	64 x 8	Stores the luma Y component of the display memory for HDMI output buffer
U Display	64 x 8	Stores the chrominance U component of the display memory for HDMI output buffer
V Display	64 x 8	Stores the chrominance V component of the display memory for HDMI output buffer







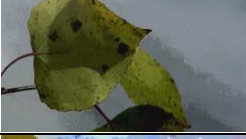

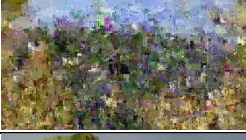
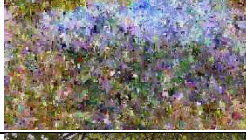












It has been shown that, the frame buffer, which is the largest memory, can tolerate three truncated LSBs, which provides power saving opportunities for the hardware design. Table 10 lists the results with LSB truncation in different video memories using the video system shown in Figure 8. The standard video sequence *aspen\_1080p.y4m* [42], which has a wide range of plain macroblock percentage was 20.90%; the maximum and minimum were 50.89% at frame #367 and 3.03% at frame #113, respectively. The video was encoded with the following ffmpeg [45] command:

```
ffmpeg -i aspen_1080p.y4m -profile:v baseline -pixel_format  
yuv420p -level 3.1 -framerate 30 -preset 1 -cavlc 1 -pix_fmt  
yuv420p aspen_1080p.264
```

### **Influence of Video Content on Viewer Experience in the Presence of Hardware Noise**

Traditionally, hardware designers have used PSNR for evaluating video quality, which has been recently shown to be insufficient to demonstrate the viewer's experience [57, 58]. PSNR does not encompass the necessary information to hardware designers about viewer experience, due to the fact that key influencing factors for viewer experience, such as video content and environment conditions, are not included in PSNR [58]. This work aims to find a better method to analyze videos in a quantitative way that will also be useful to hardware researchers. This process begins with the PSNR metric being used to describe video quality. New insights to the traditional PSNR metric are introduced with the introduction of content-aware information. This new form of information allows for gracefully scaling the video quality with enhanced energy efficiency of hardware.

Table 10. Results of videos with different LSBs truncated in different memories

<i>Memories</i>	<b>Number of LSBs Truncated</b>	<i>PSNR</i> (Max MB %)	<i>PSNR</i> (Min MB %)	<b>Max Plain MB % Frame</b>	<b>Min Plain MB % Frame</b>
Original video frames without any truncation	-	-	-		
<i>Chroma Level Cb</i>	5 LSBs truncated	43.0 dB	31.1 dB		
<i>Chroma Level Cr</i>	5 LSBs truncated	36.3 dB	27.0 dB		
<i>Luminosity Level</i>	5 LSBs truncated	18.0 dB	16.7 dB		
<i>Restructured Neighboring</i>	1 LSB truncated	13.7 dB	11.0 dB		
<i>Prediction Mode</i>	1 LSB truncated	23.0 dB	19.5 dB		
<i>Motion Vector X</i>	1 LSB truncated	29.2 dB	13.5 dB		
<i>Motion Vector Y</i>	1 LSB truncated	29.1 dB	13.3 dB		
<i>Reference Macroblock (MB)</i>	5 LSBs truncated	42.8 dB	32.8 dB		
<i>Frame Buffer</i>	3 LSBs truncated	44 dB	25.5 dB		
<i>YUV Display</i>	2 LSBs truncated in each vector	11.8 dB	13.2 dB		





### ***Traditional PSNR Metric***

Although the PSNR metric, described in equation (7), is simple for hardware designers to understand, it does not truly capture the effect that errors have on a user's perception of the video. To show the lack of complete information the PSNR provides in terms of user perception, the bit truncation technique is applied to two videos and the PSNR values are calculated for 1 to 4 LSBs truncated within the luma data (i.e. the luminance channel, or Y component in raw YUV videos). The bit truncation technique is adopted to enable energy-quality adaption, which is due to the following two reasons: (i) bit truncation causes blurring in videos, which is similar to the "banding distortion" in the codec-algorithm field, and the video degradation is much less noticeable to viewers as compared to other low-power techniques such as voltage scaling [30] and (ii) the power/energy savings with bit truncation is much more significant than other low power techniques such as voltage scaling [56].

Table 11 shows two videos that were downloaded from Google's recently released YouTube-8M dataset [44], which is the largest multi-label video dataset to date. To maintain a short and consistent size label for all included YouTube video samples, the video tag will be included, which is the last portion of the full URL address. These tags are used as a unique key that points to the corresponding YouTube video. For example, the video tag EFv2FvnlLao can be used to locate the original video sample on YouTube using the following URL: <https://www.youtube.com/watch?v=EFv2FvnlLao>. As observed in Table 10, using the bit truncation technique, the PSNR value is reduced by approximately 7dB, on average, for each additional truncated LSB.

Table 11. PSNR of different videos with bit truncated applied

Video output quality with 3 LSBs truncated	# LSBs truncated	PSNR (dB)
	<b>Video #1</b> (video tag: EFv2FvnLao)	
	1	52.868
	2	44.433
	<b>3</b>	<b>37.490</b>
	4	30.985
	<b>Video #2</b> (video tag: FNlpA4FME-8)	
	1	52.741
	2	44.461
	<b>3</b>	<b>37.693</b>
	4	31.154

Both videos have very similar PSNR values with the same number of LSBs truncated, but the visual quality is significantly different. As compared to video #1 (video tag: EFv2FvnLao), the “banding distortion” of video #2 (video tag: FNlpA4FME-8) is much more noticeable to viewers. Accordingly, the traditional PSNR video quality metric cannot correlate well with viewer experience, and the video-content properties, such as the texture/motion characteristics, significantly affect a viewer’s experience. Due to these factors, this work introduces video content information to study viewer experience. Specifically, the recently developed macroblock characterization is adapted to analyze the pixel-luminance values’ variance [59], as described in the next subsection.

### *Video Macroblock Variance Analysis*

The macroblock variance analysis is typically conducted during the video pre-processing stage when encoding videos [59, 60]. The analysis used in this work adopts the defined calculation for determining whether a given macroblock is considered to be plain or textured,

which avoids introducing significant computational overhead. This calculation is based on the variance of pixel luminance values of a given macroblock and is defined as [59]

$$V_{MB} = \sum_{i=0}^{15} \sum_{j=0}^{15} (P(i,j) - \rho_{MB})^2 \gg 8$$

$$MB = \begin{cases} \textit{Plain} & \textit{if } (V_{MB} \leq Th_{Low}) \\ \textit{Textured} & \textit{Else} \end{cases} \quad (10)$$

where  $\rho_{MB}$  and  $V_{MB}$  are the average luminance and variance of luminance values in a given macroblock (denoted  $MB$  in equation 14), respectively. The value used for  $Th_{Low}$  was 1.25 as was determined in [60] through the use of regression analysis. For analysis purposes, this  $Th_{Low}$  value is an arbitrary number used to define the plain macroblock percentages in the model design process. This macroblock characterization can be calculated during the encoding process and transmitted as metadata in the video bit stream. Currently, an embedded system implementation is used to calculate the average plain macroblock calculation. To minimize computational overhead, a single, averaged plain macroblock percentage is calculated that represents an entire sample. However, it is possible to calculate a per frame macroblock percentage for videos that change scenes frequently for dynamic adaptation. Two benchmark videos, Akiyo and News, were initially retrieved from [42]; these videos contain static backgrounds with a low amount of motion from the reporter(s) in the videos. Both videos display low plain macroblock percentages when analyzed. 32 video samples were collected with similar broadcasting characteristics from the YouTube-8M dataset [44] and the minimum, maximum, median, and average plain macroblock percentages were calculated for each sample. Figure 9 displays two video samples with similar PSNR values, but varying plain macroblock percentages for 2 LSBs truncated. The distribution of plain macroblocks and the resulting banding distortion effect are visualized within the red blocks of Figure 9.

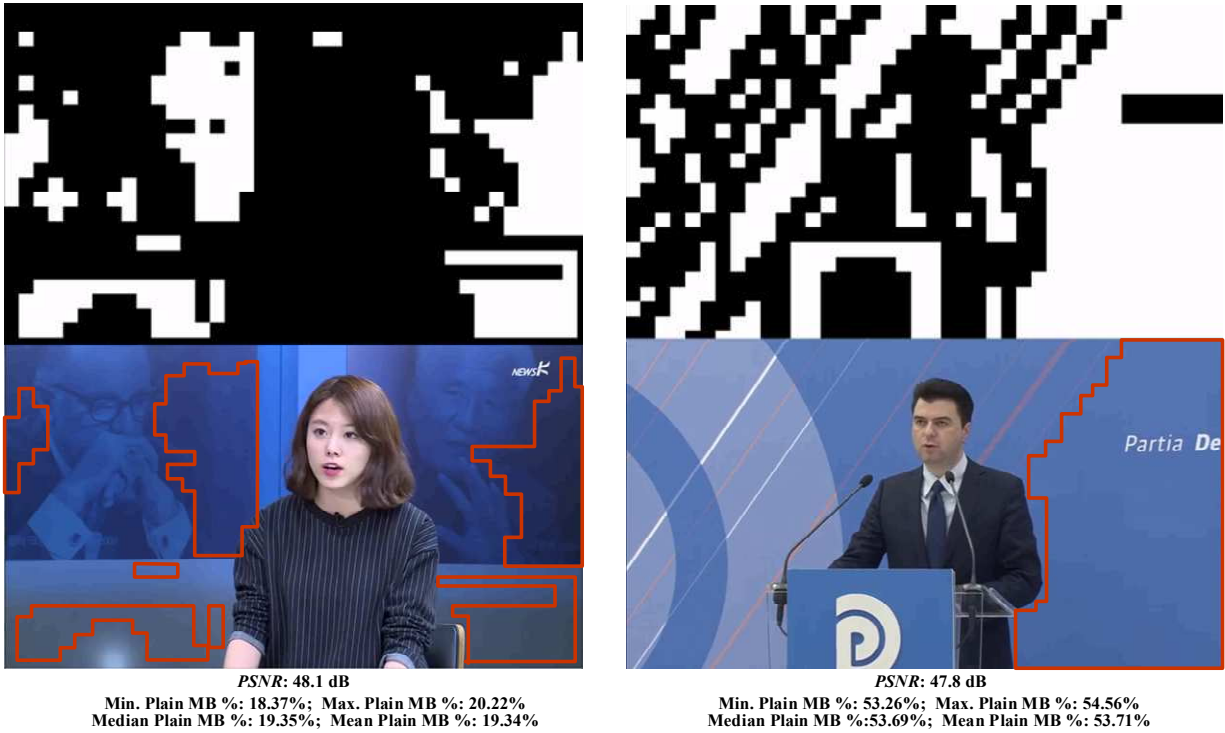


Figure 9. Plain macroblock visualization and video output comparison (white = plain)

An important observation is that a noticeable relationship exists between the banding distortion and the plain macroblocks; videos with large amounts of plain macroblocks, especially where the plain macroblocks are dense, tend to have decreased visual quality to viewers. Accordingly, this relationship is utilized to develop a content-adaptive model to predict the acceptable number of truncated LSBs for different videos. Specifically, to minimize the computational overhead, the average plain macroblock percentage per video frame is used and the focus of the video samples is videos with low-motion and a stationary camera, or containing a reporter in the analysis.

### Modeling Process

To determine the acceptable number of LSBs to truncate for different videos, subjective video testing was conducted and based on the collected data, two models were developed using a

decision tree and logistic regression methods. For the initial study, only the luma (Y) component is considered when truncating LSBs.

### **Subjective Testing Procedure for Data Collection**

Two sets of subjective video studies were conducted to collect viewers' feedback. Within each of the studies designed for subjective analysis of truncation techniques, participants were asked to view multiple versions of the same video. The testing procedure follows guidelines from the ITU [61] and uses the Degradation Category Rating (DCR) method [57], which is also known as the Double Stimulus Impairment Scale (DSIS). The participants were asked to watch both the original video and a truncated version of the video and then score them from 1 to 5 based on their opinion of the quality (imperceptible-5, perceptible but not annoying-4, slightly annoying-3, annoying-2, very annoying-1). We used an average score of 4.0 or higher as the target for acceptable video quality [61]. The first (second) of two studies contained 10 (13) participants who were each asked to view 7 (9) individual videos from the 34 total videos.

With these averages scores for different amounts of LSBs truncated, the video samples were split into different regions. Based on this, models that connect the average plain macroblock percentages to the number of LSBs that can be truncated were developed.

### **Modeling Process**

*1) Decision Tree Model:* From the initial subjective studies, the goal is to model the correlations between the calculated average plain macroblock percentages and the largest amount of LSBs that can be truncated for a given PSNR that will maintain an acceptable video quality. Figure 10 displays the video samples' average plain macroblock percentage and how many bits can be truncated based on the minimum acceptable impairment score of 4.0. The number of acceptable bits to be truncated were determined through 2 sets of subjective trials (i.e. Subjective

#1 and Subjective #2 in Figure 10) and are indicated by 1T for 1 LSB truncated, 2T for 2 LSBs truncated, and 3T for 3 LSBs truncated.

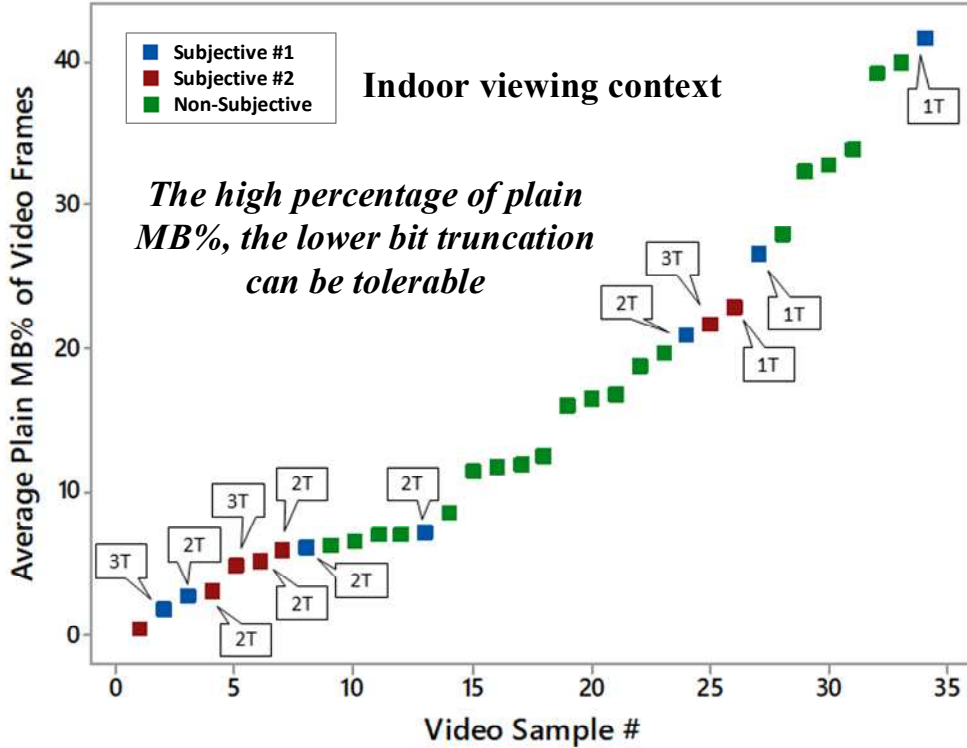


Figure 10. Acceptable truncated bits based on subjective feedback

From these preliminary results, an inverse relationship can be seen between plain macroblock percentage and acceptable number of LSBs to truncate. With this knowledge and subjective data gathered from participants, a decision tree model using the Classification Learner tool in MATLAB is developed, as shown in Figure 11. By traversing the tree from the top to the bottom based on the plain macroblock percentages, the number of truncated LSBs can be obtained for different videos. It is worthy to mention that the majority of videos from the YouTube-8M dataset have plain macroblock percentages above 1.96405% (see Figure 11) and therefore the number of videos with the decision for 3 LSB truncation is much less than that of 1 LSB and 2 LSB truncation.

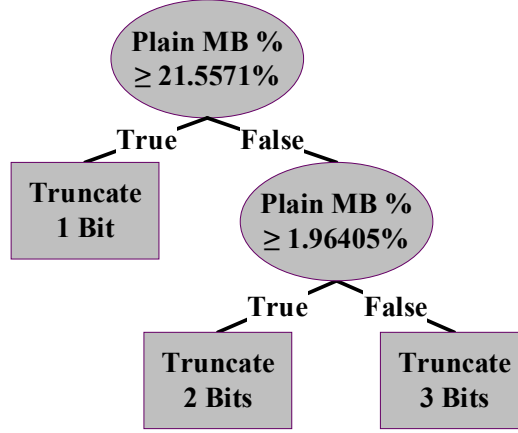


Figure 11. Developed decision tree model for bit truncation

2) *Logistic Regression Model*: In the model development process, the widely-applied statistical modeling method, logistic regression, was also considered, which is represented by

$$\begin{cases} \pi_i = \pi_3 \exp(\beta_{i0} + \beta_{i1}x), & i = 1,2 \\ \pi_3 = \frac{1}{1 + \exp(\beta_{10} + \beta_{11}x) + \exp(\beta_{20} + \beta_{21}x)} \end{cases} \quad (11)$$

where  $\pi_i := P\{Y = i|x\}$  indicates the probability that the number of truncated LSBs is  $i$  for a given average plain MB percentage of  $x$ . MATLAB was used to fit the  $\hat{\beta}$  coefficients and resulted in the following values:  $\hat{\beta}_{10} = -1.6636$ ,  $\hat{\beta}_{11} = 12.7929$ ,  $\hat{\beta}_{20} = 1.4408$ ,  $\hat{\beta}_{21} = 1.0497$ . The corresponding p-values were 0.243, 0.103, 0.111, 0.881, respectively. This implies that all four coefficients are not significant in the regression under a 5% significance level. By observing the data, one can clearly see that this is due to noise.

If a user chooses a video as satisfactory that is truncated by  $k$  LSBs, then he/she will be satisfied by the same video truncated by  $k'$  LSBs where  $0 < k' < k$ . The difference between  $k$  LSB and  $k'$  LSB truncation is the energy efficiency that can be enabled; the efficiency is higher for  $k$  LSB truncations. To this end, the ordinal logistic regression is used for modeling, which yields

$$\ln\left(\frac{\pi_1}{\pi_2+\pi_3}\right) = \beta_{10} + \beta_1 x \quad (12)$$

$$\ln\left(\frac{\pi_1}{\pi_2+\pi_3}\right) = \beta_{20} + \beta_1 x \quad (13)$$

$$\pi_1 + \pi_2 + \pi_3 = 1 \quad (14)$$

Solving (12), (13), and (14), leads to

$$\pi_1 = \frac{\exp(\beta_{10}+\beta_1 x)}{1+\exp(\beta_{10}+\beta_1 x)}, \pi_2 = \frac{1}{1+\exp(\beta_{10}+\beta_1 x)} - \frac{1}{1+\exp(\beta_{20}+\beta_1 x)}, \text{ and } \pi_3 = \frac{1}{1+\exp(\beta_{20}+\beta_1 x)} \quad (15)$$

MATLAB is used to fit the ordinal coefficients resulting in  $\hat{\beta} = [\hat{\beta}_{10}, \hat{\beta}_{20}, \hat{\beta}_1] = [-2.8322, 0.9856, 9.7783]$ , with p-values  $\mathbf{p} = [0.0039, 0.1710, 0.0156]$ , respectively. With this ordinal logistic regression, only  $\beta_{20}$  is not significant under a 5% significance level and the result is much better than the previous case using the standard logistic regression. Table 12 lists the ordinal logistic regression results. From this table it can be seen that there is no decision for 3 LSB truncation based on the ordinal logistic regression model. This is mainly because very few videos with 3 truncated LSBs are considered acceptable by the participants; also, most of the video testing results with 3 LSB truncation are considered to be noisy data. When the plain macroblock percentage ( $x$ ) is 0.28504 (i.e., 28.504%),  $P\{1 \text{ LSB truncated}\} = P\{2 \text{ LSBs truncated}\} = 0.4888$ . So, if  $x > 28.504\%$ , 1 LSB is truncated; otherwise, 2 LSBs are truncated.

Table 12. Results of ordinal logistic regression

$x$	$P$ {1 LSB truncated}	$P$ { 2 LSBs truncated }	$P$ { 3 LSBs truncated }	<i>Decision for LSB truncation</i>
0.05	0.0876	0.7261	0.1863	2 LSBs
0.10	0.1354	0.7415	0.1231	2 LSBs
0.15	0.2034	0.7174	0.0793	2 LSBs
0.20	0.2939	0.6560	0.0502	2 LSBs
0.25	0.4043	0.5643	0.0314	2 LSBs
<b>0.28504</b>	<b>0.4888</b>	<b>0.4888</b>	<b>0.0224</b>	<b>2 LSBs</b>
0.30	0.5253	0.4552	0.0195	1 LSB
0.35	0.6434	0.3446	0.0120	1 LSB
0.40	0.7463	0.2463	0.0074	1 LSB
0.45	0.8275	0.1679	0.0046	1 LSB
0.50	0.8866	0.1105	0.0028	1 LSB
0.55	0.9273	0.0710	0.0017	1 LSB
0.6	0.9541	0.0448	0.0011	1 LSB



## Quality Optimized Bit Truncation Design

This section will cover a newly proposed viewer-aware bit truncation technique, which has less visual quality degradation with the same number of LSBs truncated. Based on the developed bit truncation technique and models, an energy-quality scalable memory with content adaptation is implemented.

### Quality Optimized Bit Truncation

Bit truncation can adjust the video data's bit-depth by disabling LSBs to enable power savings and has been applied widely in low power hardware design [55, 56]. Here, viewer awareness is introduced to the hardware design process and is used to develop a new hardware implementation scheme for bit truncation with a minimized effect on the viewer's experience.

Suppose that the goal is to truncate the lowest  $t$  LSBs of each luma (Y) byte. For a given video, the true numerical value can be calculated for the truncated bits. However, if all videos in general are considered, the true (decimal) value of these truncated  $t$  LSBs should be considered a random variable. These truncated  $t$  LSBs may express any decimal numbers among  $0, 1, 2, \dots, 2^t-1$ , because there is not any general prior knowledge that works for all videos. A crucial question is as follows: what value should be set/given after the true value of these lowest  $t$  bits are truncated? A natural and intuitive method is to make them all zeros. For example, if the true value of a byte is  $10101110_2$  and three bits are truncated, then the byte's value after truncation is  $10101000_2$ . Setting the truncated bits as zeros has been widely adopted by designers [55, 56]. However, in the following proposition, it is shown that this value is not the best for minimizing the expected mean square error,  $E(MSE)$ .

**Proposition 1.** Suppose that the lowest  $t$  LSBs of a byte are truncated. Without losing generality, it is assumed that the true value of these bits is evenly distributed. Then, the best value for these  $t$  truncated bits, in terms of minimizing  $E(MSE)$ , is  $10 \dots 0_2$  (with  $t - 1$  zeros).

*Proof.* Let random variable  $Y$  indicate the true numerical value which is expressed by the truncated  $t$  LSBs.  $Y$  is evenly distributed, therefore the probability mass function (pmf) for  $Y$  is shown in Table 13.

Table 13. Probability mass function for random variable  $Y$

$Y =$	0	1	2	...	$2^t - 1$
probability	$1/2^t$	$1/2^t$	$1/2^t$	...	$1/2^t$

Let  $x$  be the targeted (decimal) value that is set for these truncated LSBs. The aim is to minimize  $E(MSE)$ , namely to minimize

$$f(x) = \frac{1}{2^t} [(x - 0)^2 + (x - 1)^2 + \dots + (x - (2^t - 1))^2] \quad (16)$$

Let

$$0 = f'(x) = \frac{1}{2^{t-1}} [x + (x - 1) + \dots + (x - (2^t - 1))] \Rightarrow x = 2^{t-1} - \frac{1}{2} \quad (17)$$

Because  $x$  is an integer,  $x$  can take the value  $x = 2^{t-1} = 10 \dots 0_2$  (with  $t - 1$  zeros) or the value  $x = 2^{t-1} - 1 = 01 \dots 1_2$  (with  $t - 1$  ones). For the results presented herein, the value  $10 \dots 0_2$  was used; however, the value  $01 \dots 1_2$  could also be used. For a hardware implementation, which value is better would depend on factors such as power requirements of using  $10 \dots 0_2$  vs.  $01 \dots 1_2$ . The significance of Proposition 1 is that it shows the dependence between the value set for the truncated bits and the expected MSE and that it gives the best value, in general. To verify this proposition, 2,000 unique videos, representing 100,000 individual frames, were randomly selected from the YouTube-8M dataset [44]. As illustrated in Figure 12, setting the truncated bits

to be  $10 \dots 0_2$  (with  $t - 1$  zeros) can enable much higher  $PSNR$  values, thereby providing a better viewing experience for the same videos in the same surroundings.

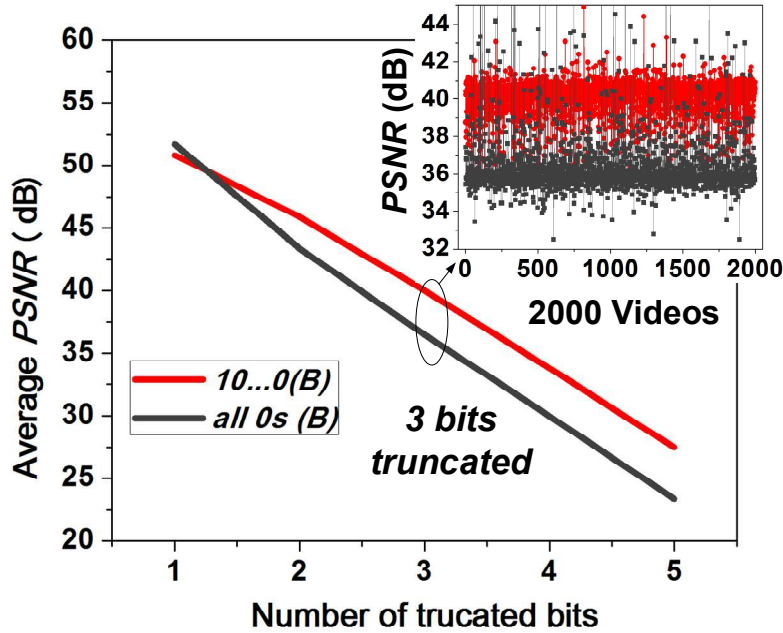


Figure 12. Average PSNR values using two different truncation techniques

### Content-Adaptation Video Memory Design

Figure 13 (a) shows the architecture of the proposed viewer-aware dynamic bit-truncation memory with  $512 \text{ words} \times 64 \text{ bits}$ , which contains 32kb of 6T SRAM bit-cells. To enable viewer-aware bit truncation for LSBs, two different bit-line conditioning circuits are applied to the memory. The normal bit-line conditioning circuits have a pre-charge unit, write driver, and sense amplifier, and they are connected to the 4 MSBs in a byte; the remaining bit-lines contain extra circuitry to enable bit truncation, and are applied to the 4 LSBs in a byte as shown in Figure 13 (b).

The truncation controller is shown in Figure 13 (c).  $\phi 1$  and  $\phi 2$  are signals generated from peripheral circuitry based on the clock signal.  $\phi 1$  controls read and write operations depending on which period it is in;  $\phi 2$  controls the pre-charging circuitry of the memory. The *sense* signal

only turns on for a very short time at the end of the reading operation in order to reduce the power consumption during the read operation. The truncation process is controlled by three external signals.  $trunc\_en$  controls whether the truncation function is on, and the other two signals,  $B<0>$  and  $B<1>$ , determine how many bits to truncate.  $t_1$  and  $t_2$  are generated from  $B<0>$  and  $B<1>$  through two decoders. The decoder for  $t_1$  is a normal 2-to-4 decoder. A special 2-to-4 truncation control decoder is applied for generating  $t_2$ , and the truth table is also shown in Figure 13 (c). When  $t_1$  and  $t_2$  are both 0s, the normal operations are applied; whenever  $t_1$  is 1, the pre-charging, writing, and reading operations are suspended; on the basis of  $t_1$  being 1, if  $t_2$  is 1 then the output will be 0, otherwise the output will be 1; the data pattern 01 for  $t_1$  and  $t_2$  will never appear.

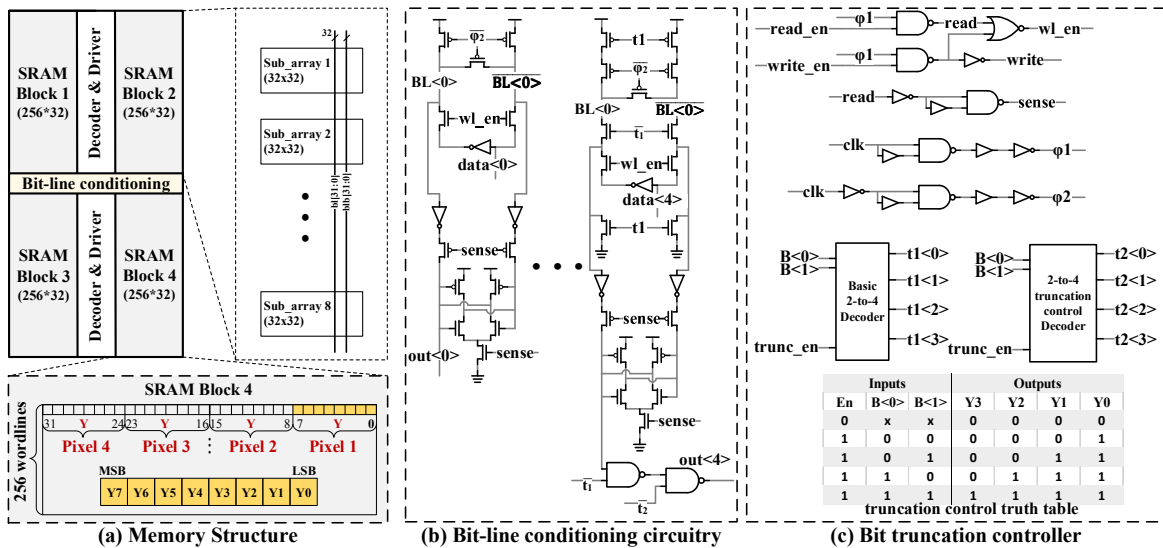


Figure 13. Content-adaptive video memory

## Experimental Results

The proposed memory is implemented using a 45nm CMOS technology [62]. In addition to hardware-level implementation and verification, psychological experiments were conducted to test the video output quality from the viewers' perspective.

## Speed

Figure 14 shows the timing diagram for the proposed memory. To test the functionality of the memory, the data:  $0xe9$ ,  $0xce$ ,  $0x62$ , and  $0x71$ , are written to the addresses:  $0x55$ ,  $0xb9$ ,  $0xce$ , and  $0x15$ , respectively, and then read out from the same addresses. For example, during a 3 bit truncation operation, the values read out are:  $0xec$ ,  $0xcc$ ,  $0x64$ , and  $0x74$ , where the last 3 LSBs for these values are  $100_2$ . The access delay of the reading operation is about 0.5 ns, which is fast enough to deliver the typical mobile video sequences (i.e. 11MHz for CIF/QCIF and 72MHz for HD720 [63]).

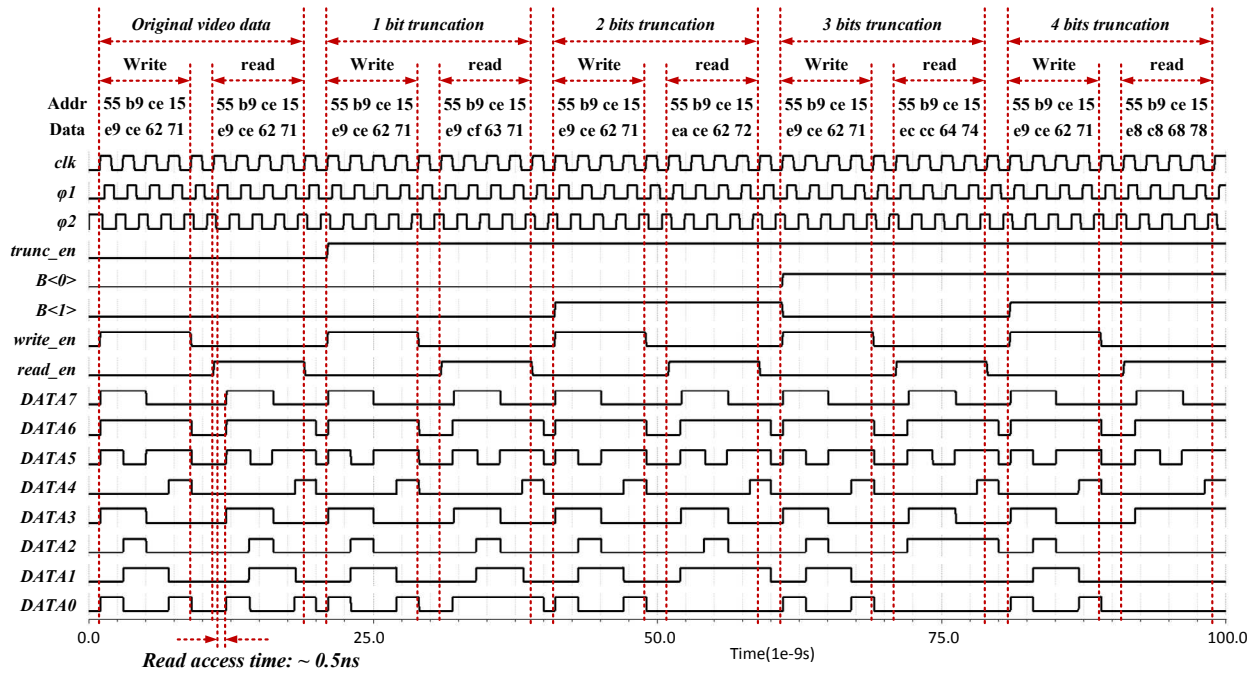


Figure 14. Timing diagram (DATA7: MSB; DATA0: LSB)

## Layout

The layout design for  $512 \text{ words} \times 64 \text{ bits}$  SRAM with viewer-aware bit truncation is shown in Figure 15. Only a few gates are added to the bit-line conditioning circuit to enable the truncation function. Also, after careful design, the decoders for truncation controlling can be fit

into the free space of the original layout, without introducing additional overhead. The proposed memory consumes only 0.32% more silicon area as compared to the traditional SRAM, which is negligible.

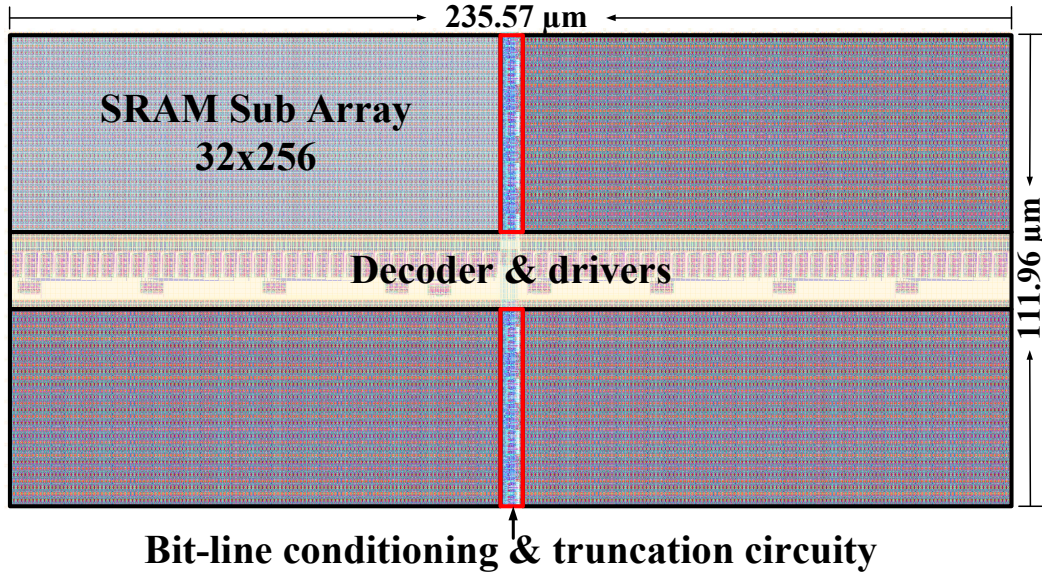


Figure 15. Physical layout design

### Power Savings

Input patterns that cover all data switching possibilities have been tested for the memory. Normal operation, and 1 to 4 LSB truncations, are simulated based on these input patterns, and the power consumption for each scenario is shown in Figure 16. As compared to normal operation, the average power consumption of reading and writing operations for 1 to 4 LSB truncations can enable 13.54%, 20.10%, 26.83%, and 33.31% power savings, respectively.

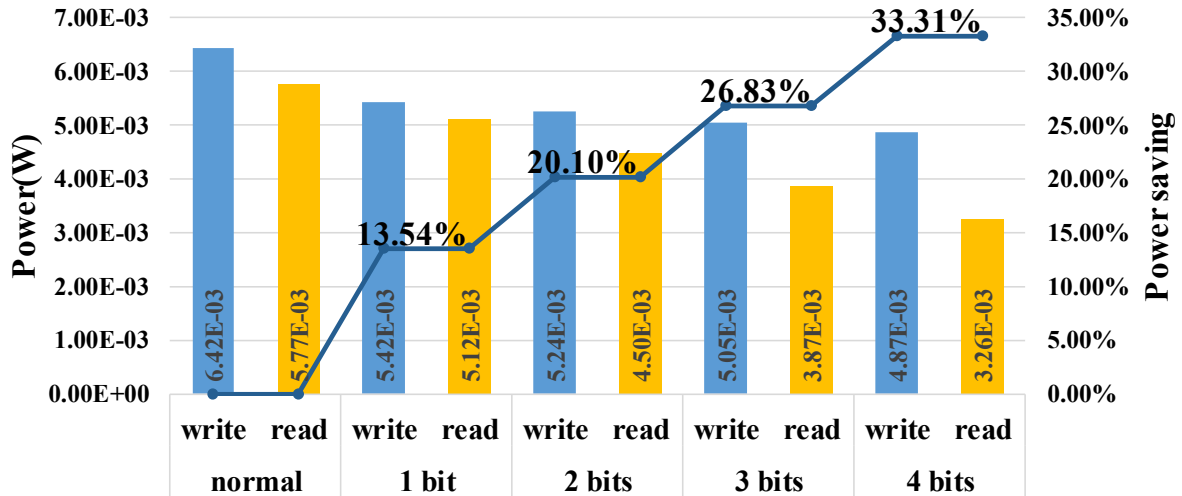


Figure 16. Power Savings

### Video Quality

Finally, in order to verify the effectiveness of the truncation technique on viewer experience, psychological experiments were conducted at the North Dakota State University Center for Visual and Cognitive Neuroscience. The psychophysical experiment setup is shown in Figure 17. The ambient illumination was provided using a rectangular array of 60 high-intensity LEDs capable of emitting a maximum of 64,000 Lux (i.e. Larson Electronics, model LEDP5W-60-D-1227-F5.15). An illumination meter (i.e. Extech model 401027) was used to accurately measure the ambient illumination of the phone used for testing, which was a Samsung Galaxy Note 4. In the experiments, the amount of illumination was adjusted for the high-intensity light source using neutral-density filters. The luminance level measured by the illumination meter was approximately 811 Lux, which is a typical indoor light level.



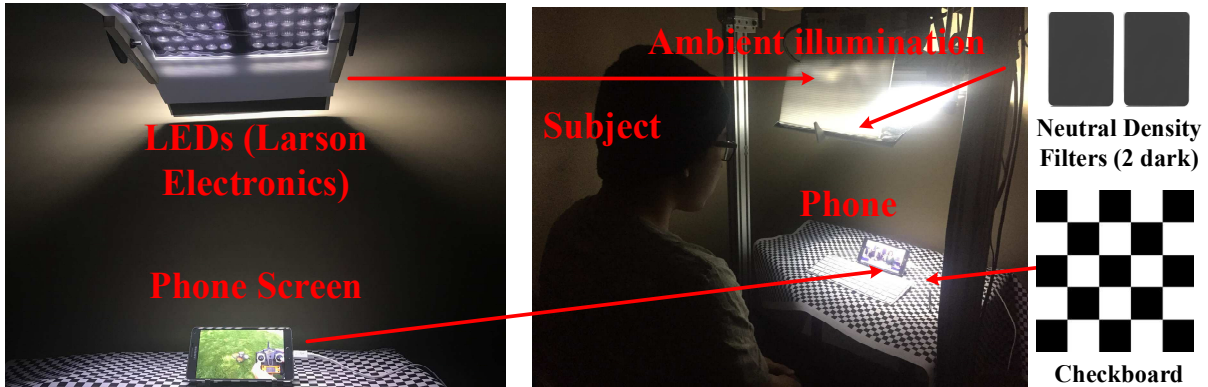


Figure 17. Psychological experiment setup

To assess the degree to which observers can accept the truncated videos as compared to the reference videos using the developed models, a total of 20 videos were collected: 10 videos were classified as having a stationary camera and 10 videos containing a reporter. Each video sample was evaluated at a single quality point encoded using a constant rate factor of 0 (i.e. lossless compression), had a  $640 \times 360$  resolution, was 10 seconds in length, and was downloaded from [44]. Based on these videos the average plain macroblock percentages were calculated and the developed models were used to predict what the expected amount of acceptable LSBs to truncate would be for different videos. Two versions of each video were created from each reference video, one with the predicted amount of acceptable bits to truncate and another with one bit beyond the predicted acceptable amount. Sequences of numbers were used to represent each video and the order they would be presented to viewers was randomized. During testing, each participant would compare a total of 40 truncated videos to their original, non-truncated version, and give their opinion of whether they would consider the video acceptable for viewing on the mobile device.

The testing results for the developed decision tree model are shown in Figure 18. The plain macroblock percentages, the number of bits truncated, and the video quality metric (VQM) [64] calculation are included for comparison among samples in Figure 18. VQM is one widely



used objective video quality metric that has been shown to have a strong correlation to the subjective viewer ratings. When calculating the VQM for each sample, we used the NTIA General Model with Full Reference Calibration, which has been standardized by both the ITU and ANSI [65]. The number of participants who considered the samples with a truncation level of one bit beyond the predicted acceptable amount is also included in Figure 18 as “Predicted + 1” under each sample. The developed decision tree model works well for nearly all videos tested. There was only one video, with tag *wF6ldXXwc4*, out of 20 videos that was considered to not be acceptable by the vast majority of participants. As shown in Figure 19 (a), this video displays banding distortion, caused by bit truncation, appearing on the reporter’s face, which is likely a viewer’s focus point. Due to this particularly noticeable distortion, viewers were less likely to accept the displayed degradation. All other samples were considered acceptable by the majority of the 15 total participants, with the lowest acceptance rate being 73% for the video with tag *2AQ6rhVhwRc*, another video with banding appearing very close to a viewer’s focus point, the kitten playing with a string in the video. When one bit beyond the predicted bit truncation value was used for the videos, only 8 of the 20 videos were considered to be acceptable by the majority of participants, and none of the video samples had unanimous acceptability.



Figure 18. Video quality testing results using the decision tree model

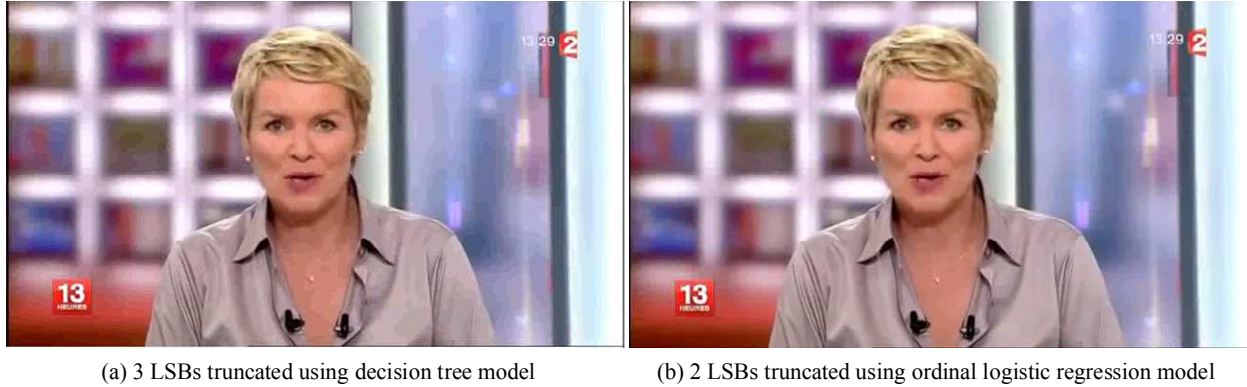


Figure 19. Output quality of video with tag *wF6lvdXXwc4*

The results using the ordinal logistic regression were further compared to the decision tree model. Those two models achieve the same prediction results for the majority of videos; only 4 out of the 20 videos are different. For the 4 videos that differ, the decision tree model predicts that 3 LSBs should be truncated and the ordinal logistic regression model predicts that only 2 LSBs should be truncated. One of those 3 videos is the video with tag *wF6lvdXXwc4*; it was the only one that was considered to be not acceptable using the decision tree model. With 2 LSBs truncated predicted by the ordinal logistic regression model, the visual quality is significantly improved, as illustrated in Figure 19 (b). For the other 3 videos (with tags *Lp3H1XOcKCE*, *dgAu\_Wsd7Fo*, and *lcVPxLFlq1c*), the visual output with 3 LSBs truncated are acceptable by the majority of participants. Particularly, for the video with tag *dgAu\_Wsd7Fo*, all of the participants said it was acceptable. From the above analysis, it can be concluded that as compared to the decision tree model, the ordinal logistic regression model is a more conservative model, which can avoid the worst video quality degradation case, but it may lose energy optimization opportunities for some videos. Another interesting observation that was made during the video testing process is that if a viewer's focus can be detected in different videos

(e.g., mobile gaze tracker [66]), noticeable degradation can be removed from those sensitive areas of videos in the future.

### **Context-Aware Memory Concluding Remarks**

In this chapter, a video context-aware memory technique for energy-quality tradeoff using viewer perspective was presented. Based on the influence of how video content characteristics impact viewer experience, two simple, but effective models to enable hardware adaptation were developed. A new viewer-aware bit truncation technique with minimized impact on a viewer's experience was presented that introduces energy-quality adaptation to the video storage. Future investigations would include incorporating the motions of videos in the viewer experience study, as well as combining viewing luminance awareness to further enable energy-quality adaptation in different viewing surroundings.

During the hardware implementation process, a single percentage for the entire video was used in order to minimize the overhead of the design. In order to better suit the applicability and energy-quality scalability, future research could investigate the capability of calculating the macroblock percentage for each frame. This per frame calculation could allow for real-time adjustment of truncated bits at the cost of additional area overhead. Expanding the number of participants and video samples in order to create a more comprehensive model could also be used to improve the model results. Finally, further studying the relationship between the content information and the psychophysical human visual system models could be used to better understand what other metrics could be used to support the hardware design.

## **CHAPTER 4. DATA-DRIVEN INTELLIGENT EFFICIENT SYNAPTIC STORAGE FOR DEEP LEARNING<sup>3</sup>**

Nowadays, with the exponential growth of readily available information, deep learning is becoming the go to solution for various engineering problems. Autonomous vehicles, drug discovery, natural language processing, image recognition, and healthcare, are a few notable areas that have a great deal to gain from the use of these deep learning systems. The necessary deep learning models used to help perform tasks in these disciplines need continuous parameter updates during the training process, which require intensive synaptic weight read and write operations. The use of SRAM for training these types of models is critical for these systems to be able to meet performance and energy efficiency requirements. In order to achieve an optimal tradeoff between energy efficiency, area overhead, and model classification accuracy, this chapter introduces an offline data mining method in order to optimize the hardware design process. This technique will help to relieve machine learning hardware designers of the large burden of data storage in deep learning systems. A 45 nm 64 kbits SRAM for synaptic weight storage is presented that can enable 45.6% active power savings and 83.2% leakage power savings, with low area overhead (3.17%) and 0.72% loss in model classification accuracy.

### **Synaptic Storage and Memory Failure Overview**

The use of neural networks has been adopted for a wide variety of applications, including: image recognition, finance, medicine, pattern discovery, and autonomous vehicles. The memory failures in synaptic storage, specifically for artificial neural networks (ANNs) will be analyzed in this section.

---

<sup>3</sup> The material in this chapter was co-authored by Jonathon Edstrom, Yifu Gong, and Dongliang Chen. Jonathon Edstrom held the primary responsibilities of writing the simulation code, extracting data from software simulations, analyzing the data and verifying results. Yifu Gong and Dongliang Chen, provided the presented SRAM hardware design with simulation results based on the software implementation results.

## Synaptic Storage in Artificial Neural Networks

The operation of neural networks is based on the many intricately connected, biological pathways that are present within animals, such as the brain or visual cortex. Figure 20 shows the general architecture of an ANN, which contains multiple hidden layers in between an input and output layer. Each computational unit, or neuron, within the neural network has a set of signals that are passed from the input layer, through multiple hidden layers, and eventually to the output layer.

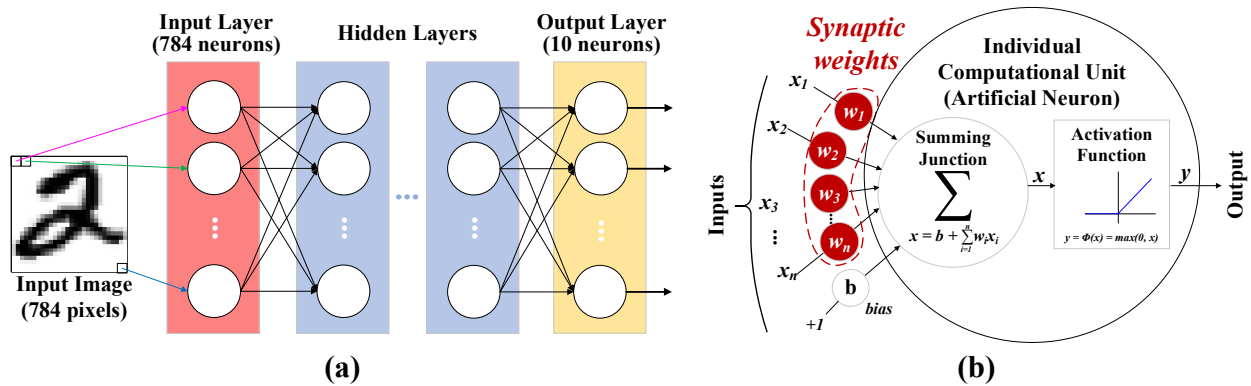


Figure 20. (a) ANN architecture; (b) Single neuron with synaptic weights

These signals begin at the input as a vector that represents the data used to train or test the neural network. The signals at each connection between layers are modified by the weight values that are associated with each pathway. Those weight values are dynamically adjusted when training the network using backpropagation, or minimizing the calculated error through an optimization method, after each batch of inputs has finished flowing through the network [67]. Accordingly, the majority of memory usage by neural networks comes from these synaptic weights that are repeatedly updated throughout the training process.

## Impact of Synaptic Memory Failures on ANN Classification Accuracy

In this subsection, the SRAM failure mechanism, and its impact on the classification accuracy of ANNs will be discussed. Among various SRAM bit cells developed by researchers, 6T and 8T are the two most widely applied SRAM bit cells. Considering the impact of technology scaling on device sizing, Lambda-based rules are used, where, one Lambda ( $1\lambda$ ) is equivalent to half the minimum feature size for a particular SRAM technology. Figure 21 shows 6T and 8T bit cells with  $3\lambda$  width using 45nm technology. The left SRAM bit cell schematic in Figure 21 (a) displays a  $3\lambda$ -6T SRAM bit cell; the pull-up transistors (PU) are minimum size, the widths of the pull-down (PD) and access transistors (AX) are  $3\lambda$ . The right SRAM bit cell schematic in Figure 21 (b) displays a  $3\lambda$ -8T SRAM bit cell, based on  $3\lambda$ -6T SRAM bit cell; two read port transistors with  $3\lambda$  widths are connected to QB. The left layout shown in Figure 2 (b) is an upsized  $9\lambda$ -6T bit cell and the right layout is a  $3\lambda$ -8T bit cell. Conventional 6T bit cells have the advantage of providing low static power dissipation, but exhibit stability problems when operating at scaled voltages. When performing a read operation, a 6T bit cell can overwrite the stored bit value with its inverted value if the voltage at QB reaches the threshold voltage of the NMOS transistor on the Q side of the bit cell [68]. By including two additional transistors, 8T bit cells separate the read and write paths so that each path can be independently optimized for its respective operation. This allows for 8T bit cells to mitigate the risk of reading incorrect data when voltage scaling. In general, 6T bit cells can be used to achieve an optimized area cost; and 8T bit cells can be used for effectively reducing the memory failures due to its decoupled read and write paths.

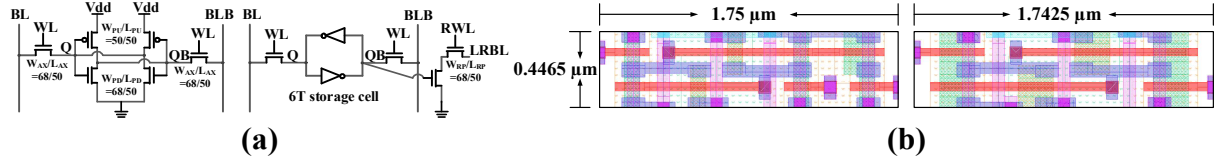


Figure 21. (a) Schematic and (b) layout design of 6T and 8T SRAM bit cells

In (b), the 6T bit cell layout on the left has been upsized ( $9\lambda$ -6T) to approximately match the size of the minimum sized 8T bit cell layout ( $3\lambda$ -8T) shown on the right. Using similar sized bit cell layouts in this way is useful to compare the failure rate due to voltage scaling between designs.

Memory failures are mainly caused by process variation in SRAM bit cells. Specifically, these types of failures are caused by threshold voltage variations ( $\sigma V_{th}$ ), which can be expressed as follows [69]:

$$\sigma V_{th} = \frac{A_{VT}}{\sqrt{WL}} \quad (18)$$

where  $A_{VT}$  is a technology dependent constant, and  $W$  and  $L$  represent the width and length of the transistor, respectively. The  $\sigma V_{th}$  for transistors with  $W$  equal to the minimum  $L_{EFF}$  for 45 nm predictive technology is 46.9mV for NMOS and 41.8mV for PMOS. From (1),  $\sigma V_{th}$  is inversely proportional to  $\sqrt{WL}$ , therefore, as  $W$  and  $L$  increase, the deviation of  $V_{th}$ , the voltage threshold of the SRAM bit cell, is reduced.

Based on a Monte-Carlo simulation with 1,000,000 trials, the read and write failure rates were estimated at the five separate process corners, including: “ss” (slow NMOS and slow PMOS), “sf” (slow NMOS and fast PMOS), “fs” (fast NMOS and slow PMOS), “ff” (fast NMOS and PMOS), and “tt” (typical NMOS and typical PMOS). At the same voltage, the failure rate decreases as the size of the SRAM bit cell increases. The failure rates of  $3\lambda$ -6T and  $3\lambda$ -8T SRAM bit cells in different process corners are shown in Figure 22 (a) and (b), respectively. Figure 22 (c) displays the failure rates of 6T bit cells with various sizes, from  $3\lambda$  to  $16\lambda$ , and an 8T bit cell with  $3\lambda$  width, in their worst process corners (i.e. “fs” for 6T bit cells and “sf” for 8T bit cell).



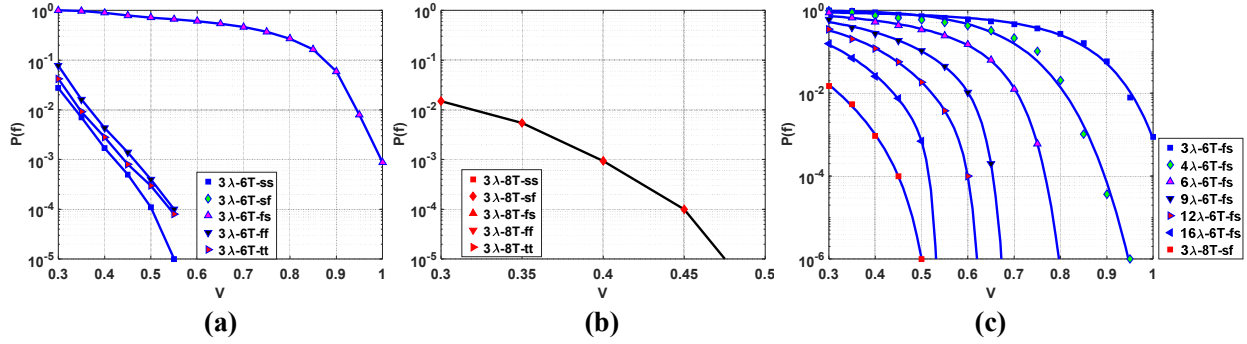


Figure 22. 45nm SRAM bit cell failure rates based on  $V_{dd}$  voltage scaling

The failure rates of the 8T bit cell for the simulated 45nm technology are much smaller than  $10^{-5}$  and are not displayed in Figure 22. However, the additional two transistors in 8T SRAM bit cells induce a 27.7% area overhead, as compared to 6T SRAM bit cells.

Due to the significant area overhead 8T SRAM bit cells induce on the overall design, it is useful to compare an upsized 6T SRAM bit cell with similar area, which was found to have  $9\lambda$  width for 45nm technology. Based on Figure 22, with similar layout area,  $3\lambda$ -8T bit cells have significantly lower failure rate than  $9\lambda$ -6T bit cells at the same operating voltage. Therefore,  $3\lambda$ -8T bit cells are used for the sign bit and the second MSB,  $W_{30}$ , in order to minimize the failure rate of these two bits.

A neural network was simulated on the widely used digit recognition dataset – MNIST [70]. A large range of bit cell failure rates were tested in order to obtain the largest possible failure rate the memory could endure for  $\sim 1\%$  degradation to the network’s classification accuracy. It was found that a maximum memory failure rate of  $10^{-5}$  allows for less than 1% loss to the accuracy. In order to achieve minimum area overhead,  $4\lambda$ -6T sizing was adopted to be the baseline synaptic memory at 1.0V in the memory analysis.

## **Data Characteristics of Synaptic Storage**

In this subsection, the data characteristics for synaptic storage will be analyzed. While fixed point synaptic weight representations allow for reduced resource requirements as compared to floating point representations, they introduce loss of precision and added design complexity. The designed neural networks use the IEEE 754 single precision floating point representation, which has been used widely in previous works [71, 72]. An investigation of data contribution, bit switching statistics, and association/correlation characteristics of synaptic weights will be discussed.

### **Data Contribution Characteristics**

First, the contribution of each bit within the synaptic weights to output precision is analyzed. Based on the MNIST handwritten digits benchmark [70], a typical ANN was created that contained two hidden layers. This model was used to test the contribution of different bits within the separate layers of the neural network architecture. Using the Python library Keras [73], a variety of neural network configurations were developed for testing and verification. Data for the neural network model was stored in memory using the IEEE single precision floating point format as shown in Figure 23. In this floating point format, the sign bit indicates whether the number is positive or negative, the exponent bits determine the exponent of the number modified by a bias, and the significand represents the significant digits of the number following the binary point (i.e., mantissa = 1.significand, such that the mantissa's leading 1 is implied, not stored as part of the 32-bit floating point number).

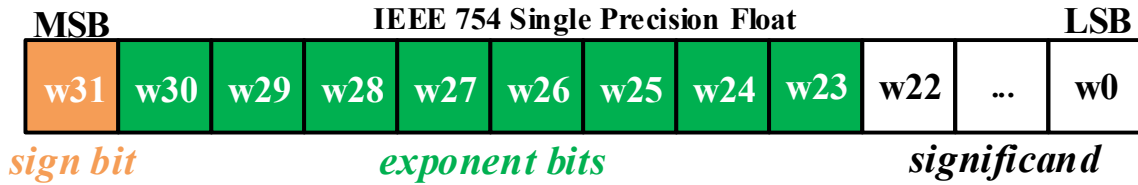


Figure 23. IEEE 754 Single Precision Floating Point Representation

A random number generator and user specified failure rate for memory bit cells were used to test each of the 32 bit positions for their impact on training the entire system. Failures were injected one layer at a time to explore the data contribution to the entire system. The network model designed for analyzing bit injection training degradation is described in Table 14.

Table 14. ANN architecture and configurations

Dataset	# of Neurons per Hidden Layer	Epochs	Batch Size	Failure Rate
MNIST	20	20	128	1.0 (100%)

The classification accuracy of a neural network, with 100% failure rate for a specific bit position can be seen in Figure 24. The testing accuracy of this neural network with no faults injected was approximately 96%. As shown, the 7 MSBs of the 8 exponent bits have a noticeable impact on the overall test accuracy regardless of which layer they are in; whereas a sign bit error only severely degrades the network accuracy when occurring on a transition from the second hidden layer to the output (i.e., sign bit errors from the input to the first hidden layer and between hidden layers have much less impact on overall accuracy).

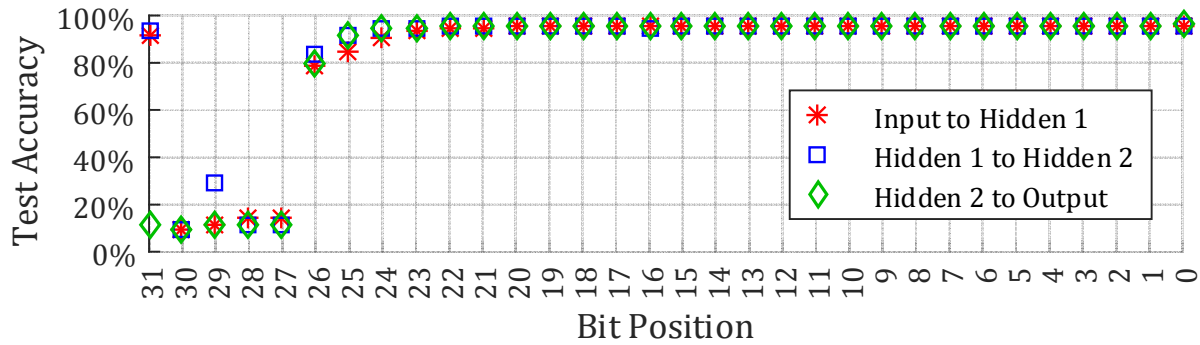


Figure 24. Influence of synaptic weight bit position on ANN classification accuracy

### Data Switching Characteristics

A further analysis of the individual bits within the synaptic weights reveal useful information about the bit switching characteristics. In an ANN model with 5 neurons per hidden layer, the weights and bias values that are stored between each of the layers were extracted between each update. Figure 25 shows the average bit switching percentage for each bit position. Based on these results, the MSBs of each weight, especially: the sign bit, all exponent bits, and the 5 MSBs of the significand, regardless of which layer they were present in, all tended to have very low switching characteristics (< 10%) between each batch update in the memory.

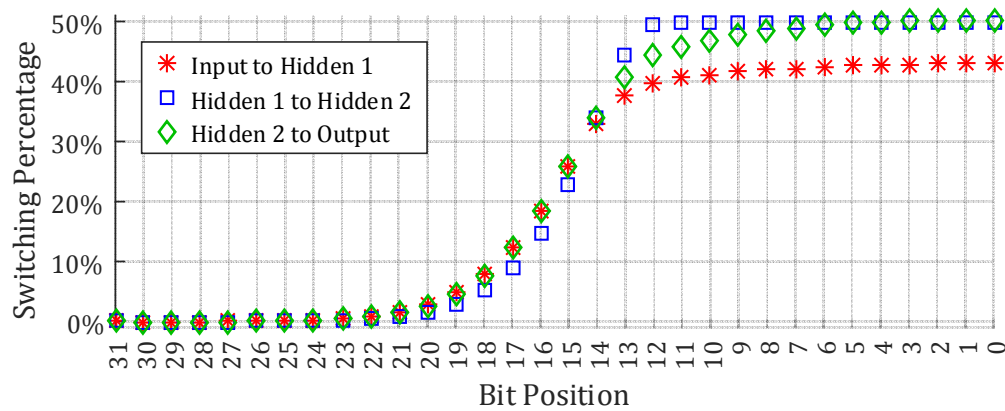


Figure 25. MNIST average bit switching percentage of each bit position

Since the first eight MSBs have the largest impact on the optimization calculations during the training process and also exhibit low switching probabilities, the use of association and/or correlation relationships between bits have the potential to be exploited through the use of offline association rule mining techniques.

### Data Association / Correlation Characteristics

Association rule mining was introduced in 1993 to discover relationships between different variables, called items, in a dataset or database [74]. A complete dataset is made up of many transactions where each transaction contains a set of items. Each item can be associated with a binary attribute, 0 or 1, that is used to distinguish if that item is present or not in the corresponding transaction. This type of data organization is illustrated in Figure 26. Each resulting rule, generated from the association rule mining process, is an implication of the form  $X \rightarrow Y$ , where X and Y are disjoint sets of, or individual, items. Each rule is also accompanied by collected statistics from the dataset called support and confidence values. The support value for a set of items is the proportion of transactions in the dataset that contains such set of items. The confidence value for an association rule,  $X \rightarrow Y$ , is the proportion of transactions that contain X which also contain Y, or the conditional probability  $P(Y | X)$ .

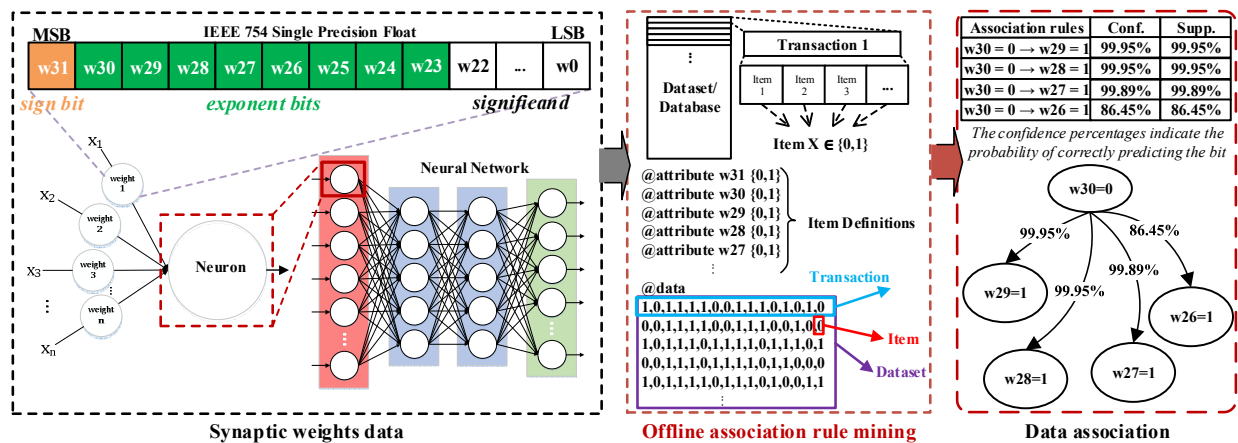


Figure 26. Offline data-mining assisted synaptic data relationships study

Association data mining techniques were used to analyze the extracted MSBs from the MNIST [70] and Abalone [75] benchmarks, to study the relationship among each neuron's corresponding weights. The discovered association rules are also shown in Figure 26. From these results, within the first 8 MSBs, multiple MSBs ( $W_{29}$ - $W_{26}$ ) have strong association to the exponent MSB,  $W_{30}$  (see Figure 26). If the value of  $W_{30}$  is stored in robust memory bit cells, in the presence of memory failures in  $W_{29}$ - $W_{26}$  bits, the SRAM may achieve self-recovery based on the discovered data association or correlation rule. Since the sign bit (i.e.  $W_{31}$ ) also significantly contributes to the classification precision as discussed in the previous subsection, the sign bit and  $W_{30}$  will be protected by being stored in robust bit cells. Based on these data characteristics, a data-driven memory is proposed, which will be discussed in the next section.

### **Proposed Data-Driven Synaptic Memory**

This section provides a detailed explanation of the proposed hardware design. The simulation results for performance, area cost, and power efficiency are discussed. An overall comparison of the memory design compared against the state-of-the-art is also displayed.

#### **Implementation**

Figure 27 shows the architecture of the proposed synaptic memory with  $256 \text{ words} \times 256$  bits, for a total of 64 kbits. A hierarchical readout bit line scheme (local RBL and global RBL) is applied to reduce the access time. For each 32-bit synaptic weight stored in the memory, two 8T bit cells are used to store the sign bit and exponent MSB.

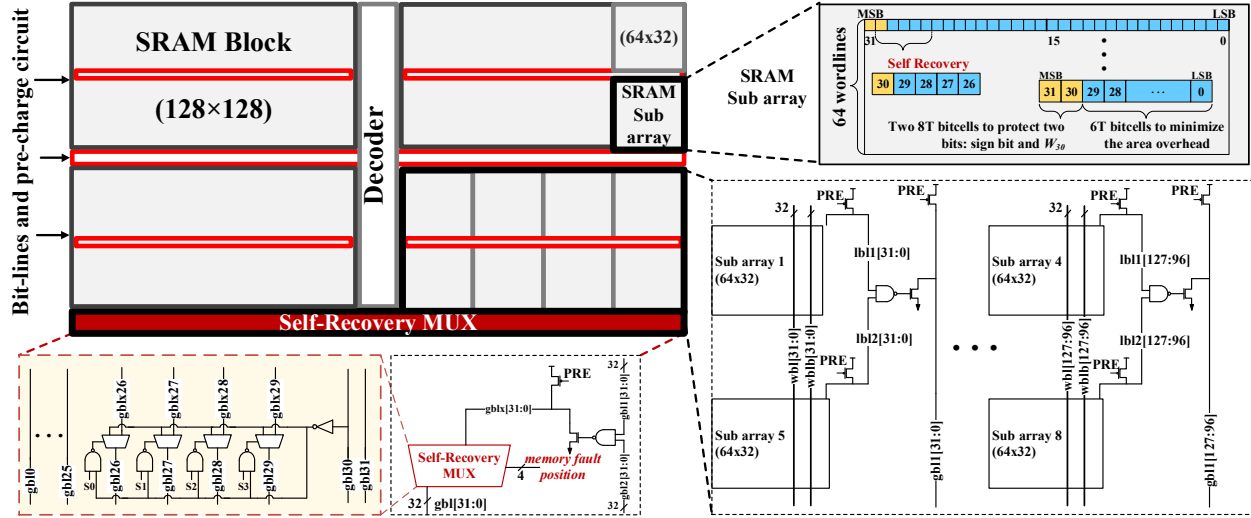


Figure 27. Data-driven efficient synaptic storage

A multiplexer-based scheme is adapted from [76] and used to implement a self-healing synaptic memory. Specifically, based on the obtained data association and correlation relationships between  $W_{29}$ - $W_{26}$  bits and  $W_{30}$ , the global bit-lines of  $W_{29}$ - $W_{26}$  are connected to a self-recovery MUX. The MUX is controlled by the pre-determined locations of the faulty bits, which are usually identified either during post-fabrication testing or Power-On Self-Test (POST). Such testing processes can also be used to track temporal degradation caused memory failures such as those caused by the aging effect. If a fault is indicated, the self-recovery process is enabled by selecting a specific data bit in the identified data relationship.

## Results

To evaluate the effectiveness of the proposed data-driven technique, a synaptic memory with  $256 \text{ words} \times 256 \text{ bits}$  is implemented in a 45nm technology [49]. Figure 28 shows the layout design. The parasitic parameters were extracted using the Cadence Virtuoso tool and included within the simulation. Based on the requirement for a worst case precision accuracy

loss of ~1%, using the proposed technique, the supply voltage can be reduced from 1V to 0.825V.

### 1) Performance

Due to the added MUXs, the read access time of the proposed synaptic storage increases from 1.154ns to 1.415ns. Based on the read access time, the calculated frequency of the proposed design is 706.7 MHz, which is fast enough to meet the high speed demands of neural network weight updates.

### 2) Layout

As discussed previously, embedded SRAMs usually occupy a large portion of area in deep learning chips, and therefore the area cost of SRAM is an important design concern. Based on the layout design of the proposed memory in Figure 28, the added self-recovery logic (MUX) results in a 3.17% area overhead. It should be noted that, the self-recovery logic is added to readout bit lines, so a memory containing more words would have even smaller area overhead.

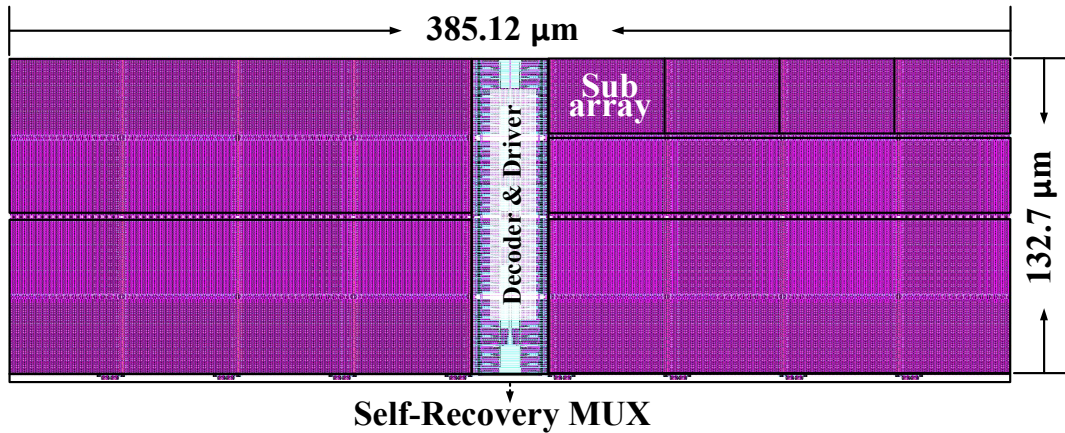


Figure 28. Layout of the proposed synaptic memory in 45 nm technology

### 3) Power Efficiency

To evaluate the power efficiency of the proposed synaptic memory, the active power and leakage power consumption was modelled as follows:



$$P_{Active} = \frac{\sum_{b=0}^{31} \sum_{i=0,1} [P_b(i) \cdot (R(i) + W(i))]}{2} \quad (19)$$

$$P_{Leak} = \sum_{d=0}^{255} \sum_{j=0}^{31} L(j) \quad (20)$$

where  $P_{total}$  is the power consumption of both the read and write operations;  $b$  is the bit number;  $I$  is the value stored in the SRAM;  $P(i)$  is the probability of a particular bit to be 0 or 1;  $R(i)$ ,  $W(i)$ , and  $L(j)$  are the read, write and leakage power consumption, respectively, based on the values  $i$  and  $j$ , which are shown in Figure 29. The bit value probabilities are extracted based on a 2 hidden layer MNIST neural network with 100 nodes per hidden layer.

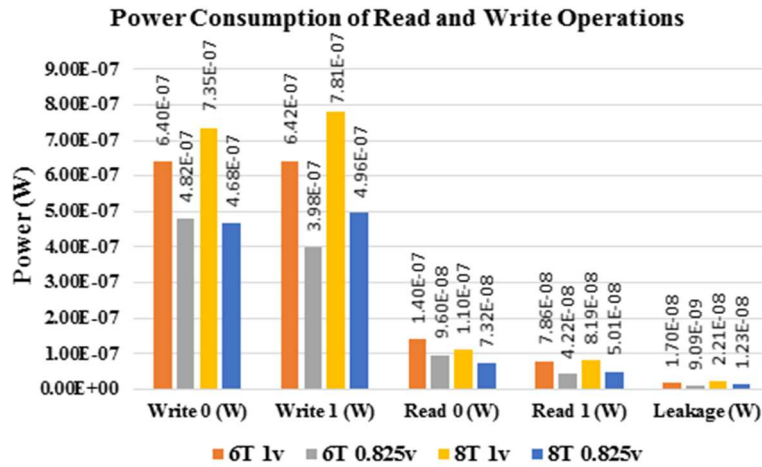


Figure 29. Power consumption of synaptic storage

Based on equations (19) and (20), the conventional SRAM at 1V consumes  $154.5\mu\text{W}$  active power and  $138.9\mu\text{W}$  leakage power, respectively. The proposed design at 0.825V consumes  $106.1\mu\text{W}$  active power and  $75.83\mu\text{W}$  leakage power, enabling 45.6% and 83.2% savings in active power and leakage power, respectively.

### Data-Driven Synaptic Memory Concluding Remarks

In this chapter, a data-driven self-correction technique was presented for neural network synaptic storage. Using data-mining discovered data characteristics, as compared to traditional memory, the proposed memory enables 45.6% and 83.2% reduction in active and leakage power

savings, respectively, with low implementation cost (3.17% area overhead) and less than 1% degradation in classification accuracy. Table 15 shows a comparison between the proposed synaptic storage system [34] against the previous state of the art design technique presented in [31]. All memories are operated at the same voltage (0.825V), indicating similar power efficiency. Based on 30 independent trials using the MNIST benchmark, the classification accuracy for a varying number of 8T cells based on [31] was evaluated, as well as the proposed data-driven technique. The averaged results of these trials compared against the fault free network, which had 96.121% classification accuracy on average, are shown in Table 15. It can be seen that the conventional 6T SRAM results in significant degradation of classification accuracy (86.321% loss) at 0.825V. In terms of [31], the larger number of 8T cells will improve the accuracy of the network, while introducing larger area overhead. For example, with 5 MSBs stored in 8T, the average loss is reduced to 1.685% with 4.015% area overhead. It can also be seen that, with similar power efficiency at 0.825V, the proposed data-driven technique exhibits a lower implementation cost with the best classification accuracy (95.401%).

Table 15. Synaptic storage comparison with existing 8T+6T hybrid design

<i>Memory Techniques</i>		<i>Average Accuracy</i>	<i>Average Loss</i>	<i>Area Overhead</i>
Traditional @1V	All 6T	96.121%	0%	0%
Traditional @0.825V	All 6T	9.8%	86.321%	0%
DATE'16 [31] @0.825V	2 MSBs 8T	92.993%	3.128%	1.606%
	3 MSBs 8T	93.120%	3.001%	2.409%
	4 MSBs 8T	94.369%	1.752%	3.212%
	5 MSBs 8T	94.436 %	1.685%	4.015%
<b>Presented Work [34] @0.825V</b>	<b>2 MSBs 8T + correction</b>	<b>95.401%</b>	<b>0.72%</b>	<b>3.171%</b>

## **CHAPTER 5. ENABLING ENERGY-EFFICIENT DIFFERENTIALLY PRIVATE EDGE INFERENCE FOR DEEP LEARNING<sup>4</sup>**

With the advent of IoT technologies and availability of a large amount of data, deep learning has been applied in a variety of applications. However, sharing personal data using IoT edge devices carries inherent risks to individual privacy. Meanwhile, the energy and memory resources needed during the inference process becomes a constraint to the resource-limited IoT edge devices. This chapter describes the process of bringing memory hardware optimization to these IoT edge devices by considering the privacy/accuracy/efficiency tradeoff in differentially efficient deep learning systems. Simulation results show that the proposed technique can enable near-threshold memory operation, and less than 1% degradation in classification accuracy.

### **Learning with Differential Privacy**

#### **Why do we need Deep Learning with Privacy?**

Privacy research has drawn attention in both industry and research communities. Large industry leaders, including: Apple, Facebook, and Google, have concluded that these types of threats can be accomplished by invasive analysts even when the data has been anonymized [77, 78, 79]. For example, in 2006 AOL released a list of 20 million web search queries which was found to have leaked the identity of a woman [80]. Similarly, Netflix held an open competition in 2006 that released a dataset that also leaked private data [81, 82]. One other area with potential privacy issues is biomedical research studies, specifically in genome wide association studies where your identity and any diseases you have could be revealed based on results included in research papers [83]. Due to privacy risks such as these, a conscious effort to reduce data leaks

---

<sup>4</sup> The material in this chapter was co-authored by Jonathon Edstrom and Hritom Das. Jonathon Edstrom held the primary responsibilities of writing the simulation code, extracting data from software simulations, analyzing the data and verifying results. Hritom Das provided the presented SRAM hardware design with simulation results based on the software implementation results.

has become of great interest, especially for companies using machine learning algorithms on collected big data.

The privacy of deep learning models, such as neural networks, have recently come into question due to weaknesses and attack models that have been previously exploited [84]. Due to high requirements of computation and storage resources, today's deep learning systems are typically built upon large, centralized data repositories. Many cloud providers also provide computing platforms and learning frameworks for model training, such as Amazon Sagemaker and Google Cloud ML Engine. Based on this centralized-training paradigm, data owners need to upload their private data to the cloud provider and they do not have control over how their private data is being used. For instance, if a deep learning model was trained on the records of patients with a certain disease, learning that an individual's record was part of the training data directly affects their privacy and it opens the door to potential misuse (e.g., exploitation for the purpose of recruitment, insurance pricing or granting loans) due to the following three potential privacy threats: (i) it is very easy for a malicious provider to steal the data if the provider has full access to the data [85]; (ii) even without full access to the data, the malicious provider can extract sensitive data from the trained models [86]; and (iii) a malicious remote user can also retrieve information of the training data by carefully querying the training models [87].

This has been demonstrated in a variety of different ways. To protect privacy, one popular type of technique is differentially private deep learning algorithms, which adds random noise to the computation so that the output does not significantly depend on any particular training sample.

## Differentially Private Deep Learning and State of the Art

Differential privacy [88] is becoming the gold standard for protecting an individual's data by introducing randomness, typically in the form of noise. The formal definition is as follows: a randomized mechanism  $M$  is considered to be  $(\epsilon, \delta)$ -differentially private if, for two adjacent inputs  $d$  and  $d'$ , it holds that  $\Pr[M(d) \in S] \leq e^\epsilon \cdot \Pr[M(d') \in S] + \delta$ , where  $S$  is any subset of outputs. The privacy cost parameter  $\epsilon$  is used to control the tradeoff between privacy and accuracy, where smaller values of  $\epsilon$  provide more privacy. The guarantee of differential privacy is: if an individual's data is used in a differentially private calculation, the probability of any result of the calculation changes by at most a factor of  $e^\epsilon$  in comparison to if that individual's data is not used in the calculation [89]. The parameter  $\delta$  is the probability of failure where the given differentially private mechanism may violate an individual's privacy. This  $\delta$  value explains the possibility of "bad events" that may result in a large loss in privacy. Specifically when training an  $(\epsilon, \delta)$ -differentially private neural network, the probability of violating the privacy,  $\delta$ , is calculated after each step for a given privacy cost,  $\epsilon$ .

Recent works have adopted the use of  $(\epsilon, \delta)$ -differential privacy in order to protect individual's data. In [90], the authors presented a technique involving an ensemble of teachers that could train on subsets of a sensitive dataset. After training, the teachers would further train a student model based on public data that was labeled using the ensemble. The student model is trained based on the noisy voting of various teachers that were trained using the model so that a stronger privacy guarantee can be enabled by the system. In [91], a method creating generative adversarial networks (GANs) that include differentially private mechanisms to provide privacy guarantees was presented. This technique for training a differentially private GAN only allows the analyst to inspect a model that already guarantees some level of differential privacy. Both the

teacher ensemble and differentially private GAN training techniques employ the use of a privacy accountant (i.e. the moments accountant), described in [92], in order to compute a tighter bound on the differential privacy.

In order to ensure differential privacy, perturbation can be introduced at various parts of the workflow, including: input, output, and objective perturbation [93]. Also, different types of noise can be added to the training and test datasets. The moments accountant shows how if the noise mechanism is Gaussian (i.e.  $\sim N(0, \sigma^2)$ ) and if the value of sigma for this noise mechanism is chosen to be:

$$\sigma = \frac{1}{\epsilon} (2 \log \frac{1.25}{\delta})^{1/2} \quad (21)$$

then the noise mechanism will satisfy  $(\epsilon, \delta)$ -differentially privacy for a given sensitivity,  $S_f$ .

Using this moments accountant technique to compute a tight bound on the privacy allows for each step in the training algorithm to result in  $(\epsilon, \delta)$ -differential privacy with respect to the lot.

The proposed system in this work uses the moments accountant to train a differentially private ConvNet model on the server (cloud) where sensitive data is used for training. By enabling the moments accountant for training, privacy can be guaranteed to some extent, but at the cost of some accuracy loss. This trained, differentially private model will then be downloaded to edge computing devices for inference tasks. A diagram of the proposed system design can be seen in Figure 30. Since inference is taken care of on the local devices, the privacy of the testing data being presented to the devices is not a big concern.

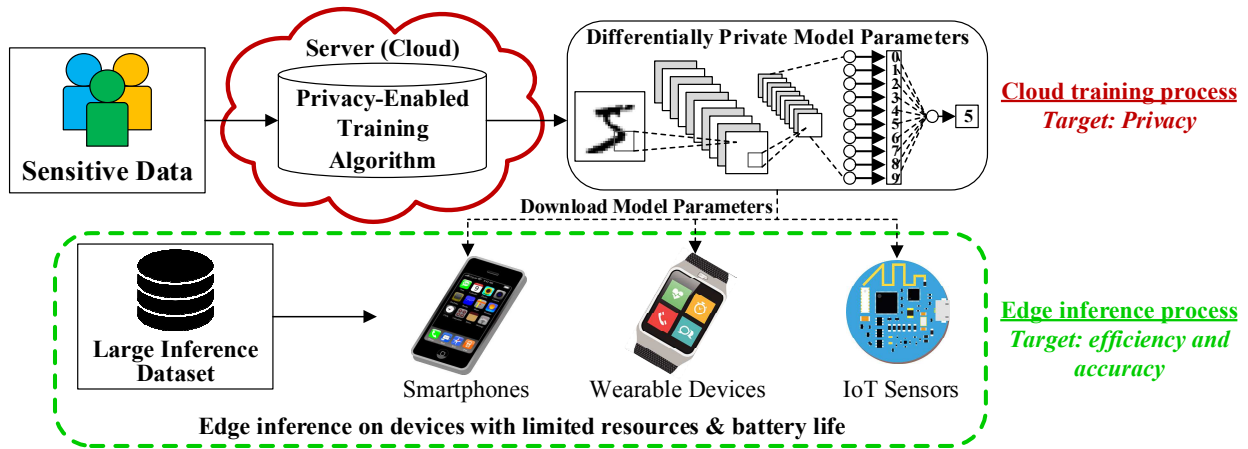


Figure 30. Proposed deep learning system with energy-efficiency/privacy/accuracy

The energy and resources needed during the inference process has become another constraint to resource-limited IoT devices. Deep learning models can take up a large portion of an embedded device’s memory space and inference tasks, especially data movements on these devices can consume the majority of the total power [94]. Software compression techniques for reducing the size of each weight in deep learning models have been introduced, such as the TensorFlow Lite API [95], which allows for 4× reduction in total model size. For hardware improvements, one of the most important issues that has been focused on is the intensive memory access of embedded IoT devices. Very recently, [96] presented a memory-based deep edge inference technique, illustrating the significance of embedded memory to edge inference. However, this technique adopted the traditional memory design, which misses out on many optimization opportunities for tradeoffs among privacy, accuracy, and efficiency.

This work aims to optimize memory design to better support differentially private deep learning algorithms in local devices. To enhance the power efficiency of memories, voltage scaling can be introduced, which causes memory failures due to process variations. Analyzing the impact of memory failures on accuracy and privacy will allow for conclusions to be drawn

for the guidelines of how to optimize the memory for privacy/efficiency/accuracy in AI applications with different requirements.

### Impact of Memory Failures in Differentially Private Deep Learning Systems

In this section, dataset quality and local memory design will be used to study the impact on accuracy of the differentially private learning process. A convolutional neural network model was defined using the TensorFlow framework [97], and will be used to gain insight on how different types and levels of noise may influence the privacy-accuracy tradeoff. The model involves using an objective perturbation through additive Gaussian noise, and uses the moments accountant [92] to compute accuracy the privacy cost after each step in the training process. The ConvNet model that was tested in this work was based on the architecture described in [96], with a single convolutional layer, and can be seen in Figure 31.

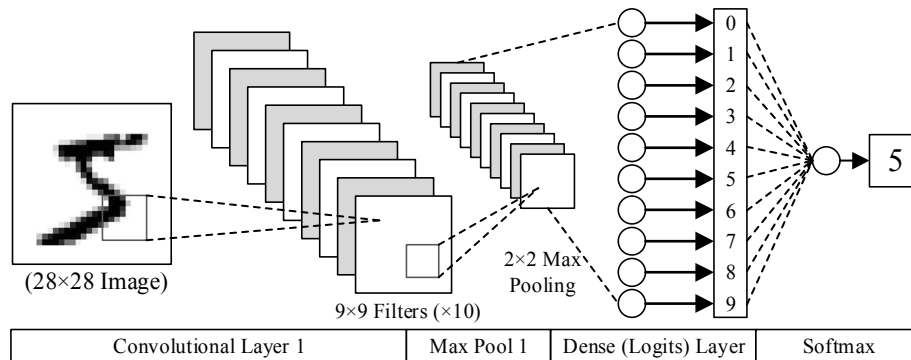


Figure 31. Differentially private convolutional neural network used for analysis

The source code used to generate models, including differential privacy calculations, can be found under [github.com/tensorflow/models](https://github.com/tensorflow/models), and was provided by the authors of [92]. The MNIST dataset [70] is used as the dataset in the neural network simulations.

### Impact of Image Quality on Classification Accuracy

In order to analyze the relationship between the quality of the test dataset and its impact on the test classification accuracy, bit level errors are injected at varying memory failure rates to



each image in the test dataset. Since the MNIST dataset consists of images, the well-known peak signal-to-noise ratio (PSNR) metric, described in equation (7), is used to evaluate quality.

Accordingly, by evaluating the PSNR values for a wide range of error injected test datasets using MNIST and comparing the test classification accuracy, we identify that the higher the image quality in the test dataset, the higher the output accuracy of the system will be overall. This relationship between PSNR and test classification accuracy is illustrated in Figure 32.

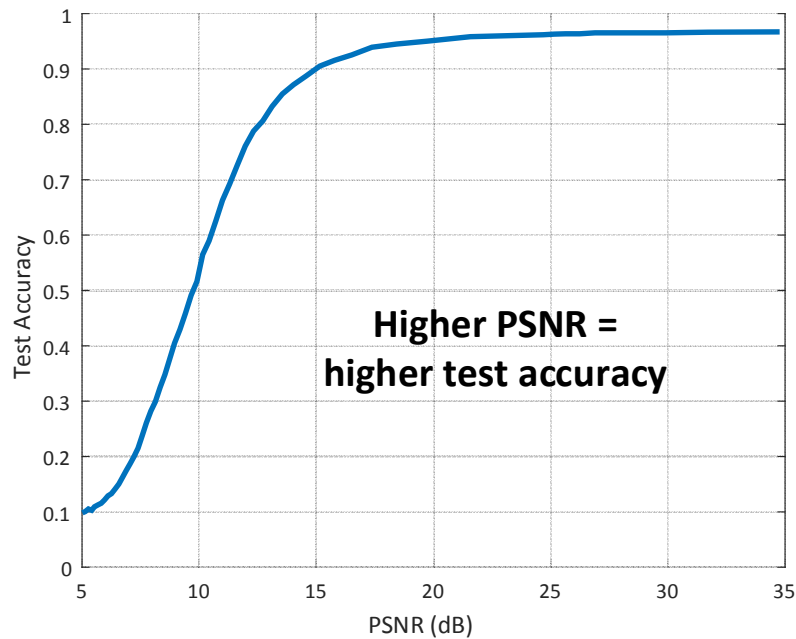


Figure 32. Influence of dataset quality on test accuracy (MNIST dataset results)

Based on this monotonically increasing behavior, if the PSNR of the test data is maximized, it is possible to achieve better system performance. In other words, during the memory design process, optimizing the hardware to maximize the quality of the dataset will improve accuracy accordingly. As shown in Figure 32, as the PSNR values of the MNIST test dataset are increased from 5dB to 15dB, test accuracy is increased from 10% to 90% using the differentially private deep learning system. It should be noted that, the samples within the MNIST datasets are images, and PSNR is an effective quality evaluation metric. If the data form

is changed, the metric will need to be adapted accordingly (e.g. SNR would be a suitable option for a sound based dataset).

### Impact of Dataset Memory in Edge Devices

The amount of Gaussian noise used during training influences how accurate the inference of the finalized model performs. Therefore, many different models were tested with varying amounts of noise (i.e. sigma values) and epsilon values with a set delta value of  $10^{-5}$ . For sigma, 4 unique noise levels were tested, specifically  $\sigma \in \mathbb{Z}^+ : 1 \leq \sigma \leq 4$ , and for each sigma value, 6 unique epsilon values, specifically  $\epsilon \in \mathbb{Z}^+ : 5 \leq \epsilon \leq 10$ , for a total of 24 combinations of values. For the MNIST dataset, the results within 1% of the floating point test accuracy for each pair that was tested can be seen in Figure 33.

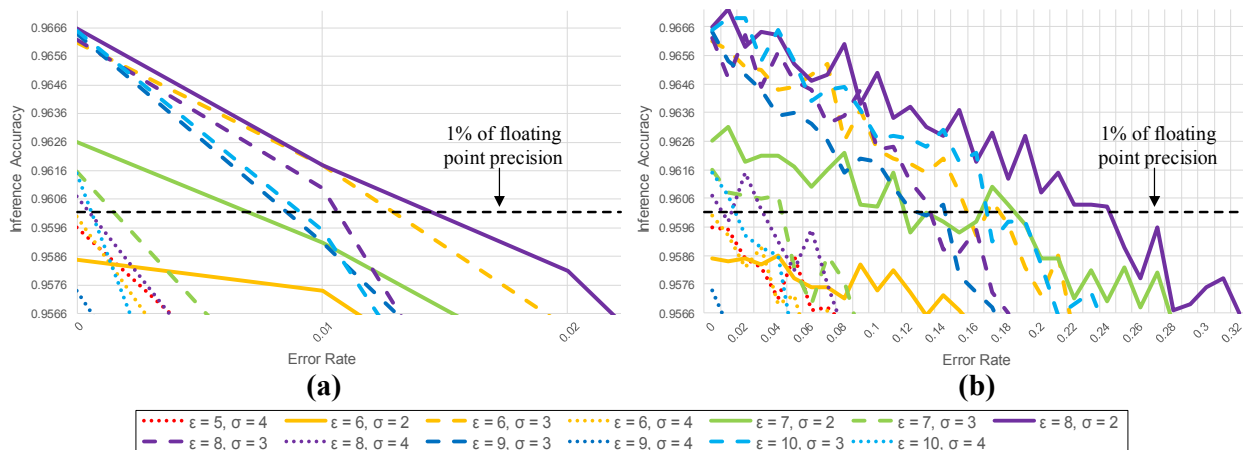


Figure 33. Impact of errors on privacy/accuracy. MSBs protected: (a) None (b) 2

The best  $(\sigma, \epsilon)$  pair (i.e. the values of sigma and epsilon that provided the best test classification accuracy) for the MNIST dataset was found to be  $\sigma = 2, \epsilon = 8$  as error rates were increased. When training using these values for the parameters, the probability of violating the privacy is recalculated after each step in the training process until the end delta value  $\delta = 10^{-5}$  to stay within a modest privacy budget [92].

One method for increasing the PSNR of the test dataset when errors are present is to protect the MSBs of the data from faults. By investigating the individual cases of protecting 1, 2, or 3 MSBs and comparing against the case without protecting any bits, it is possible to measure the influence of the MSBs on the test classification accuracy. Figure 34 displays the test classification accuracy of the  $\sigma = 2$ ,  $\varepsilon = 8$  differentially private ConvNet with the varying amount of MSBs protected. The protection of 2 or 3 bits has a significant influence on boosting the accuracy of the system to acceptable levels.

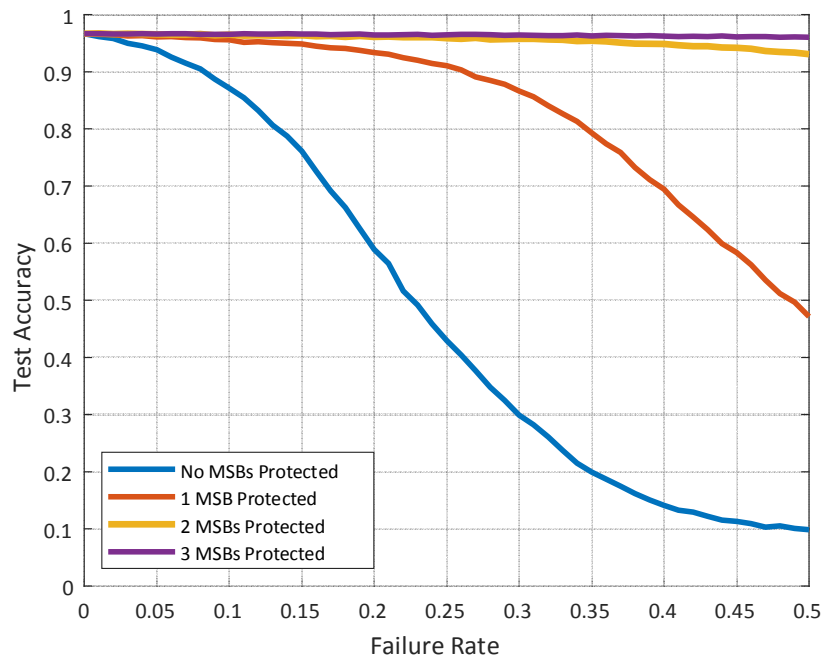


Figure 34. Impact of memory failure rate on the accuracy of the learning system

### Impact of Hardware on Privacy/Accuracy Trade-off

The impact of the memory failure on the privacy/accuracy trade-off is further examined within this subsection. From Figure 33 (a), it can be seen that the parameter  $\varepsilon$  represents the general trade-off between privacy level and accuracy of the differentially private deep learning

system. A larger value can potentially enable higher accuracy. In most cases, as the value of  $\sigma$  (i.e. the amount of noise) increases, the accuracy decreases.

By comparing (a) and (b) from Figure 33, it can be seen that for an optimized memory with MSBs protected, the accuracy/privacy tradeoff can be significantly improved. For example, if considering the specific case where  $\sigma = 2$  and  $\varepsilon = 8$ , if the failure rate is 0.23, without protection the accuracy will be  $\sim 50\%$ , an unacceptable amount. By introducing protection to 2 MSBs, at this same failure rate, the accuracy will be increased to  $>96\%$ , which is within 1% of the fault free differentially private model.

### Optimization Model based Memory Design

Based on the above analysis, developing the memory hardware to optimize the dataset quality will allow for the highest prediction accuracy. Accordingly, the problem has become an energy-quality-cost tradeoff design problem. Recently, in [98] three optimization models were presented to design the memory to provide the minimized MSE within a specific cost constraint. Model 2 in particular is useful for optimizing the dataset memory design in this work due to the quality of the dataset having a direct impact on the output accuracy. Model 2 specifically includes a method for optimizing the sizing for hybrid 8T+6T SRAM without bit cell integration cost, and is expressed in (22-25):

$$\min_x \sum_{i=1}^m \sum_{j=1}^n \sum_{k=0}^7 \sum_{l=1}^r 4^k q_{ijkl} x_{ijkl} \quad (22)$$

$$\text{s.t. } \sum_{l=1}^r x_{ijkl} \geq 1, \quad i = 1, \dots, m; j = 1, \dots, n; k = 0, \dots, 7 \quad (23)$$

$$\sum_{i=1}^m \sum_{j=1}^n \sum_{k=0}^7 \sum_{l=1}^r s_{ijkl} x_{ijkl} \leq s_{total} \quad (24)$$

$$x_{ijkl} \in \{0,1\}, \quad i = 1, \dots, m; j = 1, \dots, n; k = 0, \dots, 7 \quad (25)$$

Here,  $m$  and  $n$  are the number of pixels in a row and column in the dataset, respectively, and each pixel is composed of 8 bits. The objective function (22) is used to minimize the expected MSE of the entire dataset. Constraint (23) guarantees that for each memory bit cell, the memory designer can choose exactly one design option among the total  $r$  options. The total-area constraint, (24), assures that the total area of the design cannot exceed the limit  $s_{total}$ , where  $s_{ijkl}$  is a known parameter indicating the area cost of the  $ijk^{th}$  bit cell if it is selected to apply the  $l^{th}$  design option. The total area cost is calculated by directly summing the area cost of each bit cell, since different SRAM bit cells typically can be laid out in a mirrored fashion, and usually there is no area overhead for bit cell integration in a hybrid SRAM design [98]. Finally, constraint (25) states that all values of  $x_{ijkl}$  are binary variables.

### **Embedded Memory Design for Deep Learning**

To evaluate the effectiveness of the proposed technique, memory designs in different conditions are designed considering efficiency, accuracy, and privacy. 0.5V is used as the target voltage, considering the maximum energy efficiency enabled at near-threshold voltage. The target accuracy is 96%, which is approximately 1% less than the privacy model with no errors from voltage scaling.

For deep learning systems, memory accesses usually consume several orders of magnitude higher energy than computation, making memory performance the bottleneck for processing [94]. For example, in the deep learning IC named DianNao, the SRAM occupies 56% of the silicon area and contributes to 60% of the power consumption for the entire deep learning system [11]. Consequently, enhancing the energy efficiency of the memory is one of the key design considerations for supporting deep learning edge inference on IoT devices.

Traditional low-power memories often utilize more than 6T bit cells or use bit cell sizing to reduce memory failures induced by process variations, thereby achieving power savings at low voltages. Figure 35 shows the 6T bit cell and 8T bit cell width in a 45nm technology. 6T bit cells can achieve optimized area cost and 8T bit cells effectively reduce memory failures due to the decoupled read and write paths using two extra transistors. However, the 8T bit cell has about 9.6% area overhead compared to the 6T bit cell.

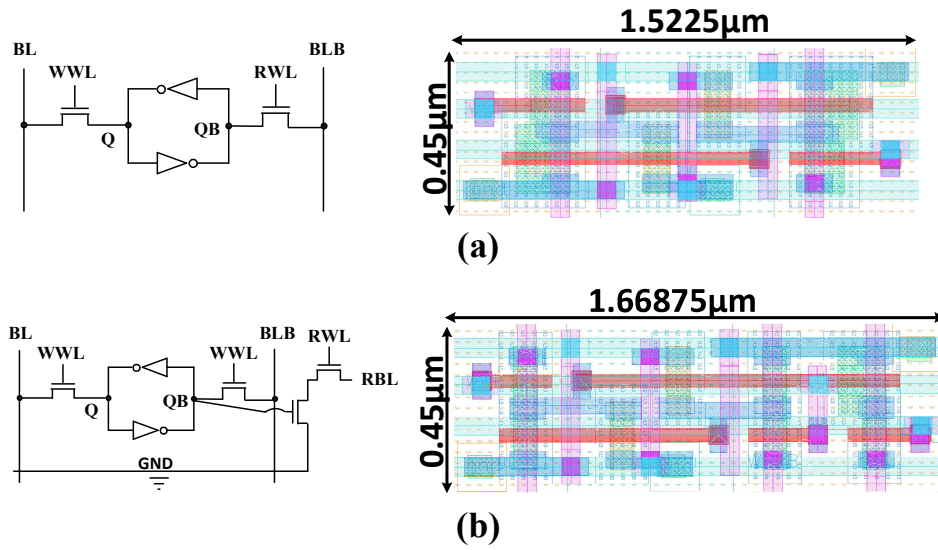


Figure 35. SRAM bit cells. Minimum sized 45nm schematic and layout: (a) 6T (b) 8T

Upsizing can also reduce the memory failure at low voltages. This is due to process variation induced threshold voltage ( $V_{th}$ ), which can be effectively reduced, and is expressed as follows:

$$\sigma V_{th} = \sigma V_{th0} \sqrt{\frac{W_{min} \cdot L_{min}}{W \cdot L}} \quad (26)$$

where  $\sigma V_{th0}$  is the standard deviation of  $V_{th}$ , and  $W$  and  $L$  represent the width and length of the transistor, respectively.  $\sigma V_{th}$  for an NMOS and PMOS transistor with  $W$  equal to the minimum  $L_{EFF}$  in the 45nm predictive technology is 46.9mV and 41.8mV, respectively. According to (26),

$\sigma V_{th}$  is inversely proportional to  $\sqrt{WL}$ , which means as the width and length increase, the deviation of  $V_{th}$  is reduced.

10,000 Monte-Carlo simulations were performed to estimate the read and write failure rates at the worst process corners for 6T and 8T: “fs” (i.e. fast NMOS and slow PMOS) for 6T and “sf” (i.e. slow NMOS and fast PMOS) for 8T bit cells, respectively. The memory bit cell data used for analysis is shown in Table 16, with  $r = r_1 + r_2 = 4 + 3 = 7$  bit cell design options. It is assumed that all pixels within the memory will use the same set of design options. Compared with the 6T options, the 8T cells require larger area, but have much lower failure rate.

Table 16. 6T and 8T bit cell design options for 45nm technology at 0.5V

Memory Type	Height ( $\mu\text{m}$ )	Width ( $\mu\text{m}$ )	Area ( $\mu\text{m}^2$ )	Area Ratio $s_k$	Failure Rate
6T: C61	0.45	1.523	0.685	1	0.3436
6T: C62	0.45	1.563	0.703	1.026	0.3074
6T: C63	0.45	1.603	0.721	1.053	0.2771
6T: C64	0.45	1.643	0.739	1.079	0.2521
8T: C81	0.45	1.663	0.751	1.096	0.00082
8T: C82	0.45	1.700	0.765	1.117	0.00009
8T: C83	0.45	1.740	0.783	1.143	0.00002

### Optimized Memory Design

The optimal values for the bit cell options based on various area constraints are shown in Table 17. The result of the proposed optimal design is compared against the traditional design. The optimized design (i.e. 8.7) has an MSE improvement of approximately 99.95% compared to the traditional 6T SRAM of an equal overall memory size (i.e. all C64 bit cells). It should be noted that if the memory area constraint is larger than 8.7, 8T bit cells could be used for all bits within each pixel.

Table 17. Optimal design results and comparison

$A_{total}$	Optimal Design									Traditional Scenario		Improvement
	$MSE_{opt.}$	$S_7$	$S_6$	$S_5$	$S_4$	$S_3$	$S_2$	$S_1$	$S_0$	$MSE_{Trd.}$	$Des_{optn.}$	
8.0	7505.94	C61	C61	C61	C61	C61	C61	C61	C61	7505.94	C61	0.00%
8.1	1889.83	C81	C61	C61	C61	C61	C61	C61	C61	7505.94	C61	74.82%
8.3	134.80	C81	C81	C81	C61	C61	C61	C61	C61	6715.15	C62	97.99%
8.5	25.11	C81	C81	C81	C81	C81	C61	C61	C61	6053.25	C63	99.59%
<b>8.7</b>	<b>2.50</b>	<b>C83</b>	<b>C83</b>	<b>C82</b>	<b>C81</b>	<b>C81</b>	<b>C81</b>	<b>C61</b>	<b>C61</b>	<b>5507.13</b>	<b>C64</b>	<b>99.95%</b>
8.9	0.7814	C83	C83	C82	C81	C81	C81	C81	C81	17.91	C81	95.64%
9.1	0.4373	C83	C83	C83	C83	C83	C83	C82	C82	1.97	C82	77.76%

### Power Consumption

The power efficiency of the optimized memory design is listed in Table 18. Operating the memory at 0.5V enables significant power savings as compared to the traditional supply voltage of 1.0V. As the total area constraint  $A_{total}$  increases, the power consumption increases as well, due to more 8T bit cells being included in the optimized design solution.

Table 18. Power consumption of optimized 45nm memory design at 0.5V

$A_{total}$	Optimal Design	Traditional Scenario		$P_{reduction}$ 0.5V (opt.) vs. 1v (Trd.)
	$P_{opt.}$ (W) at 0.5V	$P_{Trd.}$ (W) at 0.5V	$P_{Trd.}$ (W) at 1.0V	
8.0	2.07E-06	2.07E-06	9.28E-06	77.69%
8.1	2.53E-06	2.07E-06	9.28E-06	72.74%
8.3	3.01E-06	2.15E-06	1E-05	69.9%
8.5	3.50E-06	2.29E-06	1.16E-05	69.83%
8.7	3.55E-06	2.42E-06	1.41E-05	74.82%
8.9	4.09E-06	4.22E-06	1.02E-04	95.99%
9.1	3.85E-06	3.87E-06	1.02E-04	96.23%






















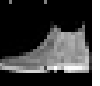







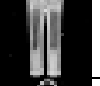










### Dataset Quality and Accuracy

The video quality and prediction accuracy based on the optimized memory is presented in Table 19. The MNIST dataset [70], which was used as the original dataset for training the CNN model, displays close to no accuracy loss (0.01%) as compared to the fault free test samples. In



order to verify this design works properly for other datasets as well, a new dataset, Fashion [99], was introduced. The Fashion dataset serves as a drop-in replacement to MNIST as it shares the same image sizes and number of classes. Training a CNN model with the same architecture on the Fashion dataset, and later testing, results in a negligible accuracy loss (0.04%) when voltage scaling to 0.5V using the optimized memory design.

Table 19. Dataset quality and accuracy for MNIST and Fashion

	No Error (1.1V)	1V Traditional	0.5V Traditional	This Work @ 0.5V
MNIST Dataset				
				
				
				
				
Test Accuracy ( $\sigma = 2, \varepsilon = 8$ )	96.7%	96.67%	42.3%	96.69%
Fashion Dataset				
				
				
				
				
Test Accuracy ( $\sigma = 2, \varepsilon = 8$ )	87.1%	87.06%	31.75%	87.07%

The results in Table 19 are based on the specific privacy level case where maximum accuracy is enabled for the MNIST dataset (i.e.  $\sigma = 2$ ,  $\epsilon = 8$ ). This privacy level still works well for the Fashion dataset, but also has potential for different tradeoff opportunities, which can be seen in Figure 36. The Fashion dataset displays higher accuracy for lower levels of  $\epsilon$ , but still performs well for varying levels of noise.

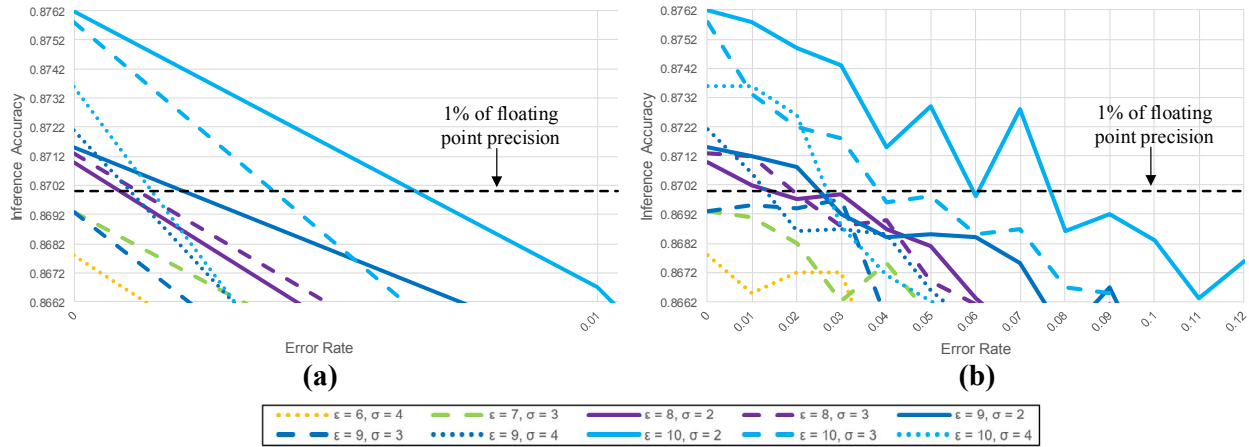


Figure 36. Verification of errors on privacy/accuracy. MSBs protected: (a) None (b) 2

### Accuracy at Different Privacy Levels

Based on CNN model simulations at varying  $\sigma$  and  $\epsilon$  values, the privacy level has a noticeable impact on the inference accuracy of the model. The MNIST and Fashion datasets were used to determine the impact of the privacy level on the inference accuracy. In general, the higher the privacy level is, the lower the test accuracy becomes. This relationship can be seen in Table 20, which includes both high and low levels of privacy for comparison of test accuracy calculations. When the test accuracy is severely degraded by the voltage scaling process, (e.g. 0.5V traditional memory design) the system does not necessarily follow the trend of higher accuracy with lower noise. However, those cases are irrelevant since they do not provide adequate quality.

As displayed in Table 20, the proposed memory design at 0.5V performs similarly to the 1V traditional design, and is capable of achieving inference accuracy within 1% of the fault free model at both low and high privacy levels.

Table 20. Impact of privacy level on test accuracy

<b>Dataset</b>	<b>Privacy Parameters</b>	<b>Privacy / Noise Level</b>	<b>1V Traditional</b>	<b>0.5V Traditional</b>	<b>This Work @ 0.5V</b>
MNIST	$\sigma = 4, \epsilon = 5$	High	95.89%	35.36%	95.91%
	$\sigma = 2, \epsilon = 10$	Low	96.52%	48.49%	96.39%
Fashion	$\sigma = 4, \epsilon = 5$	High	86.33%	27.53%	86.4%
	$\sigma = 2, \epsilon = 10$	Low	87.54%	20.14%	87.64%

### Differential Private Edge Inference Memory Concluding Remarks

Previously, in [96], an SRAM design was presented in 28nm technology that allowed for voltage scaling from the nominal voltage of 0.8V to 0.5V with 5.0× and 2.8× power reduction for leakage and memory access power savings, respectively, while allowing for an estimated  $(\epsilon, \delta)$ -differential privacy of  $(9, 10^{-5})$ . In this chapter, a memory-based deep learning system with efficiency/accuracy/privacy has been presented for IoT devices. The proposed technique can enable near-threshold memory operation at 0.5V with 74.82% power savings as compared to the traditional memory at 1.0V, and less than 1% degradation in classification accuracy with 8.75% area overhead. The presented 45nm SRAM also allows for the best inference accuracy at the  $(\epsilon, \delta)$ -differential privacy level of  $(8, 10^{-5})$ .

## CHAPTER 6. CONCLUSIONS AND FUTURE WORK

This chapter gives a brief summary of the research results presented within this dissertation, the relevance of the presented works, and how they improve over the state of the art. Potential future works for further improvements will also be discussed relating to both video and machine learning memory design.

In Chapter 2, a data-pattern enabled SRAM with self-recovery ability for big video data was presented. Based on the data patterns obtained using data-mining techniques, a simple circuit-level design technique was applied to enable memory bit cell self-recovery at near threshold voltage with a low area overhead of 7.94%. The proposed design provides 81.52% dynamic power savings and 82.45% leakage power savings as compared to the conventional nominal voltage memory. Comparing to the recently developed bit cell sizing [16], data-shifting [23], ECC [53], and data-squeezing techniques [13], the presented SRAM is capable of delivering the best video quality for the least area overhead.

In Chapter 3, a video context-aware memory technique for energy-quality tradeoff using viewer perspectives was presented. Based on the influence of how video content characteristics impact the viewer experience, two simple, but effective models to enable hardware adaptation were developed. A new viewer-aware bit-truncation technique with minimized impact on viewer experience was also implemented, which introduces optimized energy-quality adaption to the video storage. As compared to recent efficient video memory designs completed in [28, 29, 30], the newly introduced bit truncation technique provides better video quality with similar power savings.

In Chapter 4, a data-driven self-correction technique was presented for neural network synaptic storage. Using data-mining discovered data characteristics, as compared to the

traditional memory design of the same size, the proposed memory enables 45.6% and 83.2% reduction in active power savings and leakage power savings, respectively. The design has a low implementation cost of 3.17% and less than 1% degradation to the classification accuracy of the neural network architecture. As compared to the recent existing low power synaptic memory introduced in [31], the presented memory exhibits similar power efficiency with less area overhead and better classification accuracy at 95.4%.

In Chapter 5, a memory based deep learning system with efficiency/accuracy/privacy optimization was presented for IoT devices. The proposed technique can enable near-threshold operation at 0.5V with 74.82% power savings as compared to the memory operating at the nominal voltage. The design also provides less than 1% degradation to the classification accuracy with 8.75% area overhead. As compared to the recent voltage scaling technique presented in [96], the proposed memory allows for higher levels of privacy with similar power savings.

Future investigations for video memories could include: incorporating motion within videos into the viewer experience study, combining luminance with other viewing factors such as distance or movement of viewer, and calculating macroblock percentages for each frame within videos to adjust truncation levels in real-time. For machine learning memories, future work could include an extension of the proposed data-driven memory design technique to alternative representations of synaptic weights, such as fixed point or a single bit to represent each weight. For privacy based machine learning memories, exploring alternative power saving techniques (e.g. bit truncation or ECC) and their impact on the privacy/accuracy/efficiency tradeoff may lead to significant improvements to the memory design process.

## REFERENCES

- [1] E. Terzioglu, S. S. Yoon, C. Jung, R. Chaba, V. Boynapalli, M. Abu-Rahma, J. Wang, G. Nallapati, A. Thean, C. Chidambaram, M. Han, G. Yeap and M. Sani, "Low Power Embedded Memory Design - Process to System Level Considerations," in *IEEE International Conference on IC Design & Technology (ICICDT)*, Kaohsiung, May 2011.
- [2] A. Pathak, D. Sachan, H. Peta and M. Goswami, "A Modified SRAM Based Low Power Memory Design," in *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems (VLSID)*, Kolkata, Jan. 2016.
- [3] J. Gantz and D. Reinsel, "THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East - United States," February 2013. [Online]. Available: <https://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf>. [Accessed 1 May 2019].
- [4] K. Kim, "Silicon Technologies and Solutions for the Data-Driven World," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, 2012.
- [5] N. S. Rastogi, "You Charged Me All Night Long," *Slate*, 16 March 2012. [Online]. Available: <https://slate.com/technology/2012/03/is-charging-your-cell-phone-overnight-a-major-waste-of-energy.html>. [Accessed 1 May 2019].
- [6] M. A. Hoque, M. Siekkinen and J. K. Nurminen, "Energy Efficient Multimedia Streaming to Mobile Devices — A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 579-597, 2014.
- [7] Y. Benmoussa, J. Boukhobza, E. Senn and D. Benazzouz, "Energy Consumption Modeling of H.264/AVC Video Decoding for GPP and DSP," in *2013 Euromicro Conference on Digital System Design*, Los Alamitos, 2013.
- [8] D. Zhou, S. Wang, H. Sun, J. Zhou, J. Zhu, Y. Zhao, Zhou Jinjia, S. Zhang, S. Kimura, T. Yoshimura and S. Goto, "A 4Gpixel/s 8/10b H.265/HEVC Video Decoder Chip for 8K Ultra HD Applications," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, 2016.
- [9] M. E. Sinangil and A. P. Chandrakasan, "Application-Specific SRAM Design Using Output Prediction to Reduce Bit-Line Switching Activity and Statistically Gated Sense Amplifiers for Up to 1.9× Lower Energy/Access," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 107-117, 2014.
- [10] T. Marukame, K. Ueyoshi, Asai Tetsuya, M. Motomura, A. Schmid, M. Suzuki, Y. Higashi and Y. Mitani, "Error Tolerance Analysis of Deep Learning Hardware Using a Restricted Boltzmann Machine Toward Low-Power Memory Implementation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 4, pp. 462-466, 2017.

- [11] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, Salt Lake City, 2014.
- [12] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester and M. Alioto, "SRAM for Error-Tolerant Applications With Dynamic Energy-Quality Management in 28 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 5, pp. 1310-1323, May 2015.
- [13] A. Ferrerón, D. Suárez-Gracia, J. Alastruey-Benedé, T. Monreal-Arnal and P. Ibáñez, "Concertina: Squeezing in Cache Content to Operate at Near-Threshold Voltage," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 755-769, Mar. 2016.
- [14] K. Nii, M. Yabuuchi, Y. Tsukamoto, S. Ohbayashi, S. Imaoka, H. Makino, Y. Yamagami, S. Ishikura, T. Terano, T. Oashi, K. Hashimoto, A. Sebe, G. Okazaki, K. Satomi, H. Akamatsu and H. Shinohara, "A 45-nm Bulk CMOS Embedded SRAM With Improved Immunity Against Process and Temperature Variations," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 180-191, 2008.
- [15] O. Hirabayashi, A. Kawasumi, A. Suzuki, Y. Takeyama, K. Kushida, T. Sasaki, A. Katayama, G. Fukano, Y. Fujimura, T. Nakazato, Y. Shizuki, N. Kushiyama and T. Yabe, "A Process-Variation-Tolerant Dual-Power-Supply SRAM with 0.179  $\mu\text{m}^2$  Cell in 40nm CMOS using Level-Programmable Wordline Driver," in *IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, San Francisco, 2009.
- [16] J. Kwon, I. J. Chang, I. Lee, H. Park and J. Park, "Heterogeneous SRAM Cell Sizing for Low-Power H.264 Applications," *IEEE Transactions on Circuits and Systems I*, vol. 59, no. 10, pp. 2275-2284, Oct. 2012.
- [17] K. Takeda, Y. Hagihara, Y. Aimoto, M. Nomura, Y. Nakazawa, T. Ishii and H. Kobatake, "A Read-Static Noise-Margin-Free SRAM Cell for Low-VDD and High-Speed Applications," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 113-121, 2006.
- [18] T.-H. Kim, J. Liu and C. H. Kim, "A Voltage Scalable 0.26 V, 64 kb 8T SRAM with V<sub>min</sub> Lowering Techniques and Deep Sleep Mode," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 6, pp. 1785-1795, 2009.
- [19] M.-F. Chang, S.-W. Chang, P.-W. Chou and W.-C. Wu, "A 130 mV SRAM with Expanded Write and Read Margins for Subthreshold Applications," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 2, pp. 520-529, 2011.
- [20] Y.-W. Chiu, Y.-H. Hu, M.-H. Tu, J.-K. Zhao, Y.-H. Chu, S.-J. Jou and C.-T. Chuang, "40 nm Bit-Interleaving 12T Subthreshold SRAM with Data-Aware Write-Assist," *IEEE Transactions on Circuits and Systems I*, vol. 61, no. 9, pp. 2578-2585, 2014.

- [21] M. K. Qureshi and Z. Chishti, "Operating Seeded-based Caches at Ultralow Voltage with Flair," in *Proceedings of the 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Budapest, 2013.
- [22] A. Ansari, S. Feng, S. Gupta and S. Mahlke, "Archipelago: A Polymorphic Cache Design for Enabling Robust Near-Threshold Operation," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, San Antonio, 2011.
- [23] S. Ganapathy, G. Karakonstantis, A. Teman and A. Burg, "Mitigating the Impact of Faults in Unreliable Memories for Error-Resilient Applications," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, San Francisco, Jun. 2015.
- [24] I. J. Chang, D. Mohapatra and K. Roy, "A Priority-Based 6T/8T Hybrid SRAM Architecture for Aggressive Voltage Scaling in Video Applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 2, pp. 101-112, Feb. 2011.
- [25] N. Gong, S. Jiang, A. Challapalli, S. Fernandes and R. Sridhar, "Ultra-Low Voltage Split-Data-Aware Embedded SRAM for Mobile Video Applications," *IEEE Transactions on Circuits and Systems II*, vol. 59, no. 12, pp. 883-887, Dec. 2012.
- [26] S. Venkataramani, S. T. Chakradhar, K. Roy and A. Raghunathan, "Approximate Computing and the Quest for Computing Efficiency," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, 2015.
- [27] N. Gong, J. Edstrom, D. Chen and J. Wang, "Data-Pattern Enabled Self-Recovery Multimedia Storage System for Near-Threshold Computing," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Scottsdale, 2016.
- [28] D. Chen, X. Wang, J. Wang and N. Gong, "VCAS: Viewing Context Aware Power-Efficient Mobile Video Embedded Memory," in *2015 28th IEEE International System-on-Chip Conference (SOCC)*, Beijing, 2015.
- [29] J. Edstrom, D. Chen, J. Wang, H. Gu, E. A. Vazquez, M. E. McCourt and N. Gong, "Luminance-Adaptive Smart Video Storage System," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, Montreal, 2016.
- [30] D. Chen, J. Edstrom, Y. Gong, P. Gao, L. Yang, M. E. McCourt, J. Wang and N. Gong, "Viewer-Aware Intelligent Efficient Mobile Video Embedded Memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 4, pp. 684-696, 2018.
- [31] G. Srinivasan, P. Wijesinghe, S. S. Sarwar, A. Jaiswal and K. Roy, "Significance Driven Hybrid 8T-6T SRAM for Energy-Efficient Synaptic Storage in Artificial Neural Networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2016.



- [32] J. Edstrom, D. Chen, Y. Gong, J. Wang and N. Gong, "Data-Pattern Enabled Self-Recovery Low-Power Storage System for Big Video Data," *IEEE Transactions on Big Data*, vol. 5, no. 1, pp. 95-105, 2017.
- [33] J. Edstrom, Y. Gong, A. A. Haidous, B. Humphrey, M. McCourt, Y. Xu, J. Wang and N. Gong, "Content-Adaptive Memory for Viewer-Aware Energy-Quality Scable Mobile Video Systems," *IEEE Access*, 2019.
- [34] J. Edstrom, Y. Gong, D. Chen, J. Wang and N. Gong, "Data-Driven Intelligent Efficient Synaptic Storage for Deep Learning," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 12, pp. 1412-1416, 2017.
- [35] J. Edstrom, H. Das and N. Gong, "Enabling Energy-Efficient Differentially Private Edge Inference for Deep Learning," 2019.
- [36] N. Gong, S. Jiang, A. Challapalli, M. Panesar and R. Sridhar, "Variation-and-Aging Aware Low Power embedded SRAM for Multimedia Applications," in *2012 IEEE International SOC Conference (SOCC)*, Niagara Falls, Sep. 2012.
- [37] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper and N. S. Kim, "Minimizing Total Area of Low-Voltage SRAM Arrays through Joint Optimization of Cell Size, Redundancy, and ECC," in *2010 IEEE International Conference on Computer Design (ICCD)*, Amsterdam, Oct. 2010.
- [38] S. A. Pourbakhsh, X. Chen, D. Chen, X. Wang, N. Gong and J. Wang, "Sizing-Priority Based Low-Power Embedded Memory for Mobile Video Applications," in *2016 17th International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, Mar. 2016.
- [39] F. Frustaci, D. Blaauw, D. Sylvester and M. Alioto, "Better-Than-Voltage Scaling Energy Reduction in Approximate SRAMs via Bit Dropping and Bit Reuse," in *25th International Workshop on Power and Timing Modeling Optimization and Simulation (PATMOS)*, Salvador, 2015.
- [40] R. Agrawal, T. Imieliński and S. Arun, "Mining Association Rules Between Sets of Items in Large Databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington, D.C., May 1993.
- [41] "YUV Video Sequences," [Online]. Available: <http://trace.eas.asu.edu/yuv/>. [Accessed 20 March 2017].
- [42] "Xiph.org Video Test Media," [Online]. Available: <http://media.xiph.org/video/derf/>. [Accessed 20 March 2017].

- [43] "Weka," [Online]. Available: <http://www.cs.waikato.ac.nz/ml/weka/>. [Accessed 20 March 2017].
- [44] "Youtube 8M," Google, [Online]. Available: <https://research.google.com/youtube8m/>. [Accessed 4 February 2019].
- [45] "FFmpeg," [Online]. Available: <https://www.ffmpeg.org/>. [Accessed 4 February 2019].
- [46] H. Noguchi, Y. Iguchi, H. Fujiwara, Y. Morita, K. Nii, H. Kawaguchi and M. Yoshimoto, "A 10T Non-Precharge Two-Port SRAM for 74% Power Reduction in Video Processing," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Porto Alegre, Mar. 2007.
- [47] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson and S.-L. Lu, "Energy-Efficient Cache Design Using Variable-Strength Error-Correcting Codes," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, San Jose, Jun. 2011.
- [48] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S.-L. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu and D. Srivastava, "The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 846-852, Apr. 2007.
- [49] "FreePDK," [Online]. Available: <https://www.eda.ncsu.edu/wiki/FreePDK>. [Accessed 14 February 2017].
- [50] D. Zhou, S. Wang, H. Sun, J. Zhou, J. Zhu, Y. Zhao, J. Zhou, S. Zhang, S. Kimura, T. Yoshimura and S. Goto, "A 4Gpixel/s 8/10b H.265/HEVC Video Decoder Chip for 8K Ultra HD Applications," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, Feb. 2016.
- [51] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, 2004.
- [52] N. Gong, J. Wang, S. Jiang and R. Sridhar, "TM-RF: Aging Aware Power Efficient Register File Design for Modern Microprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 7, pp. 1196-1209, 2015.
- [53] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu and S.-L. Lu, "Improving Cache Lifetime Reliability at Ultra-Low Voltages," in *Proceedings of the 42nd IEEE/ACM International Symposium of Microarchitecture (MICRO)*, New York, 2009.
- [54] Q. Bin, "Osen Logic OSD10 h.264 decoder," [Online]. Available: <http://bbs.eetop.cn/viewthread.php?tid=628991>. [Accessed 15 March 2018].

- [55] D. Chen, J. Edstrom, L. Yang, M. E. McCourt, J. Wang and N. Gong, "Viewer-Aware Intelligent Efficient Mobile Video Embedded Memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 4, pp. 684-696, 2018.
- [56] F. Frustaci, D. Blaauw, D. Sylvester and M. Alioto, "Approximate SRAMs with Dynamic Energy-Quality Management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2128-2141, 2016.
- [57] L. Kerofsky, R. Vanam and Y. Reznik, "Adapting Objective Video Quality Metrics to Ambient Lighting," in *Proceedings of the 7th International Workshop on Quality of Multimedia Experience (QoMEX)*, Pylos-Nestoras, 2015.
- [58] T. Zhao, Q. Liu and C. W. Chen, "QoE in Video Transmission: A User Experience-Driven Strategy," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 285-302, 2017.
- [59] M. Shafique, B. Molkenthin and J. Henkel, "Application-Guided Power-Efficient Fault Tolerance for H.264 Context Adaptive Variable Length Coding," *IEEE Transactions on Computers*, vol. 66, no. 4, pp. 560-574, 2017.
- [60] M. Shafique, B. Molkenthin and J. Henkel, "An HVS-based Adaptive Computational Complexity Reduction Scheme for H.264/AVC video encoder using Prognostic Early Mode Exclusion," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2010.
- [61] "Methodology for the Subjective Assessment of the Quality of Television Pictures," January 2012. [Online]. Available: [https://www.itu.int/dms\\_pubrec/itu-r/rec/bt/R-REC-BT.500-13-201201-I!!PDF-E.pdf](https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.500-13-201201-I!!PDF-E.pdf). [Accessed 17 March 2019].
- [62] "FreePDK45," [Online]. Available: <https://www.eda.ncsu.edu/wiki/FreePDK45:Contents>. [Accessed 20 March 2019].
- [63] J.-S. Wang, P.-Y. Chang, T.-S. Tang, J.-W. Chen and J.-I. Guo, "Design of Subthreshold SRAMs for Energy-Efficient Quality-Scalable Video Applications," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 2, pp. 183-192, 2011.
- [64] M. H. Pinson and S. Wolf, "A New Standardized Method for Objectively Measuring Video Quality," *IEEE Transactions on Broadcasting*, vol. 50, no. 3, pp. 312-322, 2004.
- [65] "NTIA General Model (aka VQM) and Full Reference Calibration Standards," Institute for Telecommunication Sciences, [Online]. Available: <https://www.its.blrdoc.gov/resources/video-quality-research/standards/hidden-general-model.aspx>. [Accessed 20 March 2019].

- [66] Y. Feng, G. Cheung, W.-t. Tan, P. L. Callet and Y. Ji, "Low-Cost Eye Gaze Prediction System for Interactive Networked Video Streaming," *IEEE Transactions on Multimedia*, vol. 15, no. 8, pp. 1865-1879, 2013.
- [67] A. Blum, *Neural Networks in C++*, New York, NY: Wiley, 1992.
- [68] M. M. Zhang, "Performance Comparison of SRAM Cells Implemented in 6, 7, and 8-Transistor Cell Topologies," Davis, 2008.
- [69] J. A. Croon, S. Decoutere, W. Sansen and H. E. Maes, "Physical Modeling and Prediction of the Matching Properties of MOSFETs," in *Proc. 30th ESSCC*, Leuven, Belgium, 2004.
- [70] Y. LeCun, C. Cortes and C. J. Burges, "THE MNIST DATABASE of handwritten digits," 1998. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>. [Accessed 21 January 2019].
- [71] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma and G. Yang, "F-CNN: An FPGA-based framework for training Convolutional Neural Networks," in *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, London, 2016.
- [72] Y. Karri and R. Misra, "Implementation of 32 Bit Floating Point MAC Unit to Feed Weighted Inputs to Neural Networks," *International Journal of Research and Scientific Innovation (IJRSI)*, vol. 2, no. 4, pp. 40-43, 2015.
- [73] "Keras: The Python Deep Learning library," [Online]. Available: <https://keras.io/>. [Accessed 16 February 2017].
- [74] R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, Washington, DC, 1993.
- [75] "Abalone Data Set," [Online]. Available: <http://archive.ics.uci.edu/ml/datasets/Abalone>. [Accessed 3 March 2017].
- [76] N. Gong, J. Edstrom, D. Chen and J. Wang, "Data-Pattern Enabled Self-Recovery Multimedia Storage System for Near-Threshold Computing," in *IEEE 34th International Conference on Computer Design (ICCD)*, Phoenix, 2016.
- [77] A. Chin and A. Klinefelter, "Differential Privacy as a Response to the Reidentification Threat: The Facebook Advertiser Case Study," *North Carolina Law Review*, vol. 90, no. 5, 2012.
- [78] J. Tang, A. Korolova, X. Bai, X. Wang and X. Wang, "Privacy Loss in Apple's Implementation of Differential Privacy on MacOS 10.12," *ArXiv*, 2017.

- [79] Ú. Erlingsson, V. Pihur and A. Korolova, "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, Scottsdale, 2014.
- [80] M. Barbaro and T. Zeller, "A Face Is Exposed for AOL Searcher No. 4417749," *The New York Times*, New York, 2006.
- [81] D. Jackson, "The Netflix Prize: How a \$1 Million Contest Changed Binge-Watching Forever," *Thrillist.com*, 2017.
- [82] A. Narayanan and V. Shmatikov, "Robust De-anonymization of Large Sparse Datasets," in *IEEE Symposium on Security and Privacy*, Oakland, 2008.
- [83] R. Wang, Y. F. Li, X. Wang, H. Tang and X. Zhou, "Learning Your Identity and Disease from Research Papers: Information Leaks in Genome Wide Association Study," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, Chicago, 2009.
- [84] W. M. Holt, "Security and Privacy Weaknesses of Neural Networks," Provo, 2017.
- [85] C. Song, T. Ristenpart and V. Shmatikov, "Machine Learning Models that Remember Too Much," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Dallas, 2017.
- [86] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali and G. Felici, "Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers," *International Journal of Security and Networks*, vol. 10, no. 3, pp. 137-150, 2015.
- [87] R. Shokri, M. Stronati, C. Song and V. Shmatikov, "Membership Inference Attacks Against Machine Learning Models," in *IEEE Symposium on Security and Privacy*, San Jose, 2017.
- [88] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211-407, 2014.
- [89] J. Hsu, M. Gaboardi, A. Haeberlen, S. Khanna, A. Narayan, B. C. Pierce and A. Roth, "Differential Privacy: An Economic Method for Choosing Epsilon," in *IEEE 27th Computer Security Foundations Symposium*, Vienna, 2014.
- [90] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow and K. Talwar, "Semi-Supervised Knowledge Transfer for Deep Learning from Private Training Data," in *5th International Conference on Learning Representations*, Toulon, 2017.

- [91] X. Zhang, S. Ji and T. Wang, "Differentially Private Releasing via Deep Generative Model," *ArXiv*, 2018.
- [92] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar and L. Zhang, "Deep Learning with Differential Privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and*, Vienna, 2016.
- [93] A. D. Sarwate and K. Chaudhuri, "Signal Processing and Machine Learning with Differential Privacy," *IEEE Signal Processing Magazine*, pp. 86-94, September 2013.
- [94] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman and Z. Zhang, "Hardware for Machine Learning Challenges and Opportunities," in *IEEE Custom Integrated Circuits Conference*, Austin, 2017.
- [95] Google, "TensorFlow Lite," Google, 2018. [Online]. Available: <https://www.tensorflow.org/lite/>. [Accessed 3 December 2018].
- [96] L. Yang and B. Murmann, "Approximate SRAM for Energy-Efficient, Privacy-Preserving Convolutional Neural Networks," in *IEEE Computer Society Annual Symposium on VLSI*, Bochum, 2017.
- [97] Google, "TensorFlow™," Google, [Online]. Available: <https://www.tensorflow.org/>. [Accessed 11 11 2018].
- [98] Y. Xu, H. Das, Y. Gong and N. Gong, "On Mathematical Models of Optimal Video Memory Design," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 2019.
- [99] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," 28 August 2017. [Online]. Available: <https://arxiv.org/abs/1708.07747>. [Accessed 3 April 2019].

## APPENDIX A. YOUTUBE-8M VIDEO DOWNLOAD SCRIPT

```
# Python script for downloading YouTube-8M videos for running evaluation on
# Author: Jonathon Edstrom
# Department: NDSU ECE Graduate Research
# Project: Data-Pattern Enabled Self-Recovery Low-Power Storage System for Big Video Data

import random, os
from pytube import YouTube
from subprocess import call, check_output
import pandas as pd
import numpy as np

# variables
video_count = 0
label_str = None

# read the provided vocabulary.csv file from the YouTube-8M dataset
vocab_df = pd.read_csv('vocabulary.csv')

# download 10,000 unique videos from the YouTube-8M dataset
while video_count < 10000:
    try:
        full_str = random.choice(list(open('train_labels.csv')))
        url_str = full_str.split(',')[0]
        label_val = full_str.split(',')[1]
        label_num = label_val.split(' ')[0]
        label_str = vocab_df.loc[int(label_num), 'Name']
        youtube_str = 'https://www.youtube.com/watch?v=' + url_str
        print(youtube_str)
        yt = YouTube(youtube_str)
        video = yt.get('mp4', '360p')
        video.download('/home/jedstrom/data/tmp/video{}_{}.mp4'.format(video_count, label_str))
        total_frames = int(check_output(["ffprobe", "-v", "error", "-count_frames", "-select_streams", "v:0", "-show_entries", "stream=nb_read_frames", "-of", "default=nokey=1:noprint_wrappers=1", "tmp/video{}_{}.mp4".format(video_count, label_str)]))
        print('no. of frames: {}'.format(total_frames))
        randomFrame = int(random.randrange(0, total_frames-51))
        print('starting frame: {}'.format(randomFrame))
        call(["ffmpeg", "-ss", '{}'.format(randomFrame/30), "-i", "tmp/video{}_{}.mp4".format(video_count, label_str), "-vf", "scale=320:240", "-vframes", "50", "-vcodec", "rawvideo", "-pix_fmt", "yuv420p", "tmp/video{}_{}.yuv".format(video_count, label_str)])
        os.remove('tmp/video{}_{}.mp4'.format(video_count, label_str))
        video_count += 1
    except:
        if os.path.isfile('tmp/video{}_{}.mp4'.format(video_count, label_str)):
            os.remove('tmp/video{}_{}.mp4'.format(video_count, label_str))
        if os.path.isfile('tmp/video{}_{}.yuv'.format(video_count, label_str)):
            os.remove('tmp/video{}_{}.yuv'.format(video_count, label_str))
        pass
```

## APPENDIX B. YOUTUBE-8M VIDEO CLIPPED TIMING SCRIPT

```
# Python script used to download videos
# from https://research.google.com/youtube8m/
# Author: Jonathon Edstrom (2018)
# Project: Content-Adaptive Memory for Viewer-Aware
# Energy-Quality Scalable Mobile Video Systems

import random, os
from pytube import YouTube # PyTube 6.2.2
from subprocess import call, check_output

# Prompt user for YouTube video tag
url_str = raw_input("Enter the url tag for the video to download: ")

# Create video url string
youtube_str = 'https://www.youtube.com/watch?v=' + url_str

# Get video using PyTube API
yt = YouTube(youtube_str)
yt.filename = 'url={}'.format(url_str)
video = yt.get('mp4', '360p')
print(youtube_str)

# Create directory for videos to be stored and download
if not os.path.exists('videos'):
    os.makedirs('videos')
video.download('videos')

# Calculate the total number of frames and display to user
total_frames = int(check_output(["ffprobe", "-v", "error", "-count_frames", "-select_streams",
"v:0", "-show_entries", "stream=nb_read_frames", "-of", "default=nokey=1:noprint_wrappers=1",
'videos/url={}.mp4'.format(url_str)]))
print('no. of frames: {}'.format(total_frames))

# Prompt user for when they would like the clipped video to begin
skip = raw_input("How long to skip (i.e. 00:00:00): ")

# Prompt user on how many frames they would like to be in the clipped video
frames_to_process = raw_input("Frames to process: ")

# Convert the downloaded .mp4 to raw .yuv and remove the .mp4 file
call(["ffmpeg", "-ss", '{}'.format(skip), "-i", 'videos/url={}.mp4'.format(url_str), "-vf",
"scale=640:360", "-vframes", "{}".format(frames_to_process), "-vcodec", "rawvideo", "-pix_fmt",
"yuv420p", 'videos/url={}.yuv'.format(url_str)])
os.remove('videos/url={}.mp4'.format(url_str))
```



## APPENDIX C. MACROBLOCK ANALYSIS AND TRUNCATION PROGRAM

```
/*
YUV420p Macroblock Luminance Truncation Program
Jonathon Edstrom - 2017-2019
Truncates LSBs from luminance portion of YUV 4:2:0 frames with MB level analysis
Department: NDSU ECE Graduate Research
Project: Content-Adaptive Memory for Viewer-Aware Energy-Quality
        Scalable Mobile Video Systems
*/

// includes
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <math.h>
#include <stdbool.h>

// instantiate globals
FILE*   filePtr; // pointer for YUV data input/output files
FILE*   csvFilePtr; // CSV file pointer that will contain plain macroblock percentages per frame
uint8_t* buffer; // pointer for YUV data allocated memory
uint8_t* reset; // pointer value to reset the buffer to the beginning
uint8_t* compare; // pointer for YUV data to compare to (PSNR)
uint8_t* rstCompare; // pointer to beginning of comparison data (PSNR)
uint64_t filelen; // length of the input file (total bytes)
uint32_t xres; // width of the YUV video
uint32_t yres; // height of the YUV video
uint8_t plain; // # of bits to truncate in plain conditions
uint8_t textured; // # of bits to truncate in textured conditions
uint64_t lumsize; // luminance bytes per frame
uint64_t chromsize; // chrominance bytes per frame
uint64_t framecount; // total number of frames
double lowVarTh = 1.25; // low variance threshold used for determining if a MB is plain/textured
// (default value is 1.25 - from table on page 5 of "An HVS-based Adaptive Computational
// Complexity Reduction Scheme for H.264/AVC Video Encoder using Prognostic Early Mode Exclusion"
// (Shafique et. al))
double psnrValue = 0; // for later calculating the PSNR of the output video
uint8_t newTruncMethod = 1; // default to true
uint8_t outputTruncatedFile = 1; // default to true
uint8_t outputCSV = 1; // default to true
uint32_t plainMBCount = 0; // increments each time a MB is determined to be 'plain'
uint32_t totalMBCount = 0; // increments for each MB in the input video

uint8_t truncationAmount(double variance)
{
    // perform truncation based on luminance scenario and variance of MBs
    if( variance <= lowVarTh ) // plain MB
    {
        return plain;
    }
    else // textured MB
    {
        return textured;
    }
}

// helper function to truncate bits on YUV frames
uint8_t truncateBits( uint8_t byte, uint8_t bitsToTruncate )
{
    uint8_t returnByte;
    switch( bitsToTruncate ) {
        case 0:
            return byte; // don't truncate anything
            break;
        case 1:
            return ( byte & 254 );
            break;
    }
}
```

```

    case 2:
        returnByte = byte & 252;
        if(newTruncMethod == 1)
        {
            returnByte |= 2;
        }
        return returnByte;
        break;
    case 3:
        returnByte = byte & 248;
        if(newTruncMethod == 1)
        {
            returnByte |= 4;
        }
        return returnByte;
        break;
    case 4:
        returnByte = byte & 240;
        if(newTruncMethod == 1)
        {
            returnByte |= 8;
        }
        return returnByte;
        break;
    case 5:
        returnByte = byte & 224;
        if(newTruncMethod == 1)
        {
            returnByte |= 16;
        }
        return returnByte;
        break;
    case 6:
        returnByte = byte & 192;
        if(newTruncMethod == 1)
        {
            returnByte |= 32;
        }
        return returnByte;
        break;
    case 7:
        returnByte = byte & 128;
        if(newTruncMethod == 1)
        {
            returnByte |= 64;
        }
        return returnByte;
        break;
    default:
        printf( "Truncation bit value entered (%u) not valid. Exiting.\n", bitsToTruncate );
        exit( EXIT_FAILURE );
    }
}

// modified by J.E. using: https://fador.be/highlighter.php?file=psnr.c
double psnr( uint8_t *video1, uint8_t *video2 )
{
    double MSE = 0.0;
    double MSEtemp = 0.0;
    unsigned int index;

    // Calculate MSE
    for( index = 0; index < ( framecount * ( 3 * ( xres * yres ) / 2 ) ); index++ )
    {
        MSEtemp = abs( video1[index] - video2[index] );
        MSE += MSEtemp * MSEtemp;
    }
    MSE /= ( framecount * ( 3 * ( xres * yres ) / 2 ) );

    // Avoid division by zero
    if( MSE == 0 )

```

```

    {
        return 99.0;
    }
    else
    {
        return ( 10 * log10( ( 255.0 * 255.0 ) / MSE) );
    }
}

// application entry point
int main(int argc, char * argv[])
{
    printf( "\nYUV420p Luminance Byte Truncation Program Using MB Analysis (J.E. 2017-2019)\n" );
    printf( "-----\n\n" );

    if( argc != 10 ) // argc should be 9 for correct execution
    {
        // print argv[0] assuming it is the program name with the following usage hint to user
        printf( "usage: %s filename xres yres plain textured lowVarTh newTruncMethod
                outputTruncatedFile outputCSV\n\n", argv[0] );
        printf( "\tfilename: the path to the YUV file to process\n" );
        printf( "\txres: width of the YUV video in pixels\n" );
        printf( "\tyres: height of the YUV video in pixels\n" );
        printf( "\tplain: # of bits to truncate in plain Macroblocks\n" );
        printf( "\ttextured: # of bits to truncate in textured Macroblocks\n" );
        printf( "\tlowVarTh: threshold to determine if a MB is plain or textured (default:
                1.25)\n" );
        printf( "\tnewTruncMethod: (0 or 1) 0 = use old (all dropped bits to zero), 1 = use new
                truncation (median value)\n" );
        printf( "\toutputTruncatedFile: (0 or 1) Output the truncated version of the video\n" );
        printf( "\toutputCSV: (0 or 1) Output the calculated plain macroblock percentages for
                each frame to a .csv file\n\n" );
        printf( "Please correct your arguments and retry. Now Exiting...");
        exit( EXIT_FAILURE );
    }
    else // correct number of arguments
    {
        printf( "Setting things up...\n" );

        // initialize settings (global variables)
        xres = atoi( argv[2] );
        yres = atoi( argv[3] );
        plain = atoi( argv[4] );
        textured = atoi( argv[5] );
        lowVarTh = atof( argv[6] );

        // check threshold value is valid
        if( lowVarTh < 0.0f )
        {
            printf( "Invalid value for low variance threshold (lowVarTh). Exiting program.\n" );
            exit( EXIT_FAILURE );
        }
        // check to see if old truncation method should be used (all dropped bits to zero)
        newTruncMethod = atoi( argv[7] );
        // check threshold value is valid
        if( newTruncMethod != 0 && newTruncMethod != 1 )
        {
            printf( "Invalid value for using new truncation method. Exiting program.\n" );
            exit( EXIT_FAILURE );
        }

        // see if the user wants the truncatd version of the YUV file to be output
        outputTruncatedFile = atoi( argv[8] );
        if( outputTruncatedFile != 0 && outputTruncatedFile != 1 )
        {
            printf( "Invalid outputTruncatedFile value (valid: 0 or 1). Exiting program.\n" );
            exit( EXIT_FAILURE );
        }
        // see if the user wants a .csv file containing the plain macroblock percentage for each
        // frame in the video
    }
}

```

```

outputCSV = atoi( argv[9] );
if( outputCSV != 0 && outputCSV != 1 )
{
    printf( "Invalid value for outputCSV input, please input 0 or 1. Exiting program.\n" );
    exit( EXIT_FAILURE );
}

// check resolution parameters are natural numbers
if( !( xres > 0 && yres > 0 ) )
{
    printf( "Invalid value entered for the resolution parameter(s) (i.e. xres/yres).
            Exiting program.\n" );
    exit( EXIT_FAILURE );
}

// check that bit truncation values are valid
if( !( plain >= 0 && plain <= 7 && textured >= 0 && textured <= 7 ) )
{
    printf( "The value entered for one of the bit truncation parameter is not valid.
            Exiting program.\n" );
    exit( EXIT_FAILURE );
}

// create output file name string
int32_t len = strlen( argv[1] ); // get length of input file name
char filename[len];
strcpy( filename, argv[1] ); // get input file name
filename[len-4] = '\0'; // chop off the ".yuv" extension
len = len + 100; // add correct amount for output file naming
char plain_data[len];
strcpy( plain_data, filename ); // store video filename for later
strcat( plain_data, "_plainPercentagesPerFrame.csv"); // add description & file extension
if( outputCSV == 1 )
{
    // overwrite file if it exists
    csvFilePtr = fopen( plain_data, "w" );
    if ( csvFilePtr != NULL )
    {
        fputs( "", csvFilePtr );
        fclose( csvFilePtr );
    }
    else
    {
        printf( "The CSV file could not be opened for writing. Make sure the file is not in
                use. Exiting program.\n" );
        exit( EXIT_FAILURE );
    }
}
char outputstr[len];
strcpy( outputstr, filename );
strcat( outputstr, "_" );
char str[100];
strcat( outputstr, "p=" );
sprintf( str, "%u", plain );
strcat( outputstr, str );
strcat( outputstr, "_t=" );
sprintf( str, "%u", textured );
strcat( outputstr, str );
strcat( outputstr, "_lvt=" );
sprintf( str, "%f", lowVarTh );
strcat( outputstr, str );
if(newTruncMethod == 1)
{
    strcat( outputstr, "_t" );
    sprintf( str, "%u", plain );
    strcat( outputstr, str );
}
else if(newTruncMethod == 0)
{
    strcat( outputstr, "_d" );
    sprintf( str, "%u", plain );
}

```

```

    strcat( outputstr, str );
}
strcat( outputstr, ".yuv" );

// calculate Y (luminance) and UV (chrominance) byte component sizes
lumsize = xres * yres;
chromsize = lumsize / 2;

// assume argv[1] is the filename to open
// open file using "rb" = read binary file access mode
fileptr = fopen( argv[1], "rb" );

// if fopen returns a NULL pointer it failed to open the file
if( fileptr == NULL )
{
    printf( "Could not open file. Exiting program.\n" );
    exit( EXIT_FAILURE );
}
else // file opened successful -> allocate memory buffer space
{
    printf( "YUV input file opened successfully!\n" );
    fseek( fileptr, 0, SEEK_END ); // jump to end of file
    filelen = ftell( fileptr ); // get current byte offset in file
    framecount = ( 2 * filelen ) / ( xres * yres * 3 );
    printf( "Size of YUV file: %lu\n# of frames in video: %lu\n", filelen, framecount );
    rewind( fileptr ); // jump to beginning of file

    // calculate the total size of the input file in bytes
    uint64_t totalSizeOfInputFile = xres * yres * framecount * 3 / 2;

    buffer = ( uint8_t * ) malloc( filelen + 1 ); // enough memory for file + \0 (EOF)
    compare = ( uint8_t * ) malloc( filelen + 1 ); // enough memory for file + \0 (EOF)
    reset = buffer;
    rstCompare = compare;
    if( buffer == NULL || compare == NULL )
    {
        printf( "Failed to allocate memory. Exiting program.\n" );
        exit( EXIT_FAILURE );
    }
    else
    {
        printf( "Memory allocated successfully!\n" );
    }

    uint64_t bytesReadIntoBuffer = fread( buffer, 1, filelen, fileptr ); // read file

    rewind( fileptr ); // jump to beginning of file

    uint64_t bytesReadIntoCompare = fread( compare, 1, filelen, fileptr ); // read file

    if( bytesReadIntoBuffer != bytesReadIntoCompare || bytesReadIntoBuffer !=
        totalSizeOfInputFile || bytesReadIntoCompare != totalSizeOfInputFile )
    {
        printf( "Input file size error. Exiting program.\n" );
        exit( EXIT_FAILURE );
    }

    printf( "File read into memory!\n" );

    buffer = reset; // reset buffer pointer address to beginning
    compare = rstCompare; // reset compare buffer pointer address to beginning
    uint64_t index = 0; // current array index (keeps track of traversing the video)
    uint64_t frameIndex = 0; // keeps track of current frame number
    double macroblock[256]; // all luminance bytes for current macroblock

    while( index < filelen ) // loop until EOF
    {
        // report the progress of the program to the user
        if( index % ( lumsize / 2 ) == 0 )
        {
            frameIndex++;

```

```

        double percent = (double)frameIndex / (double)framecount * 100.0;
        printf( "\rFrame: %lu/%lu (%.2f%%)", frameIndex, framecount, percent );
    }

    uint8_t *currentFrame = buffer; // remember start of frame for truncating data

    // will contain how many bits to truncate for each byte in the frame
    uint8_t frameTruncationValues[xres][yres];

    // move 1D array into a 2D array (frame) of luminance data
    uint8_t luminanceFrame[xres][yres];
    for( uint32_t yIdx = 0; yIdx < yres; yIdx++ )
    {
        for( uint32_t xIdx = 0; xIdx < xres; xIdx++ )
        {
            luminanceFrame[xIdx][yIdx] = *buffer;
            buffer++;
            index++;
        }
    }

    uint32_t xpos = 0; // x position of top left pixel of current MB on current frame
    uint32_t ypos = 0; // y position of top left pixel of current MB on current frame
    bool continueLoop = true;
    // count plain MBs for the current frame for reporting to user via .csv file
    uint32_t plainMacroblocksForCurrentFrame = 0;
    uint32_t totalMBForCurrentFrame = 0; // the total MB in the current frame
    while( continueLoop )
    {
        uint32_t totalElementsInMB = 0;

        // get all bytes from the current macroblock
        uint32_t mbIndex = 0;
        for( uint32_t j = ypos; j < ypos + 16; j++ )
        {
            if( j < yres ) // ensure we are within the vertical bounds of the frame
            {
                for( uint32_t i = xpos; i < xpos + 16; i++ )
                {
                    if( i < xres ) // ensure we are within the horizontal bounds of the frame
                    {
                        macroblock[mbIndex] = (double) luminanceFrame[i][j];

                        // use calculated power function to map to real world luminance value
                        macroblock[mbIndex] = 0.0001560911143834408 * pow( macroblock[mbIndex],
                            2.628389343175764 );

                        mbIndex++;
                        totalElementsInMB++;
                    }
                }
            }
        }

        // calculate average luminance for the current macroblock
        double luminanceSum = 0;
        for( uint16_t idx = 0; idx < totalElementsInMB; idx++ )
        {
            luminanceSum += macroblock[idx];
        }
        double averageMBLuminance = luminanceSum / (double) totalElementsInMB;

        // calculate the variance of the luminance values for the current macroblock
        double variance = 0.0f;
        for( uint16_t idx = 0; idx < totalElementsInMB; idx++ )
        {
            variance += ( pow( macroblock[idx] - averageMBLuminance, 2 ) / (double)
                totalElementsInMB );
        }

        totalMBForCurrentFrame++;
    }
}

```

```

totalMBCount++;
if(variance <= lowVarTh)
{
    plainMacroblocksForCurrentFrame++;
    plainMBCount++;
}

// decide amount of bits to truncate based on luminance variance and scenario
for( uint32_t j = ypos; j < ypos + 16; j++ )
{
    if( j < yres ) // ensure we are within the vertical bounds of the frame
    {
for( uint32_t i = xpos; i < xpos + 16; i++ )
{
    if( i < xres ) // ensure we are within the horizontal bounds of the frame
    {
        frameTruncationValues[i][j] = truncationAmount(variance);
    }
}
    }

// adjust xpos and ypos for next macroblock
if(xpos + 16 < xres)
{
    xpos += 16;
}
else
{
    xpos = 0;
    if(ypos + 16 < yres)
    {
        ypos += 16;
    }
    else
    {
        continueLoop = false;
    }
}
}

// output plain macroblock percentage to a .csv file if the user wants it output
if( outputCSV )
{
    // open file using "a" = append file access mode
    csvFilePtr = fopen( plain_data, "a" );
    if ( csvFilePtr != NULL )
    {
        fprintf( csvFilePtr, "%f%%\n", plainMacroblocksForCurrentFrame * 100.0f /
            totalMBForCurrentFrame );
        fclose( csvFilePtr );
    }
    else
    {
        printf("Error opening .csv file...");
    }
}

// truncate bits for each byte in the video based on truncation values calculated
for( uint32_t yIdx = 0; yIdx < yres; yIdx++ )
{
    for( uint32_t xIdx = 0; xIdx < xres; xIdx++ )
    {
        *currentFrame = truncateBits( luminanceFrame[xIdx][yIdx],
            frameTruncationValues[xIdx][yIdx] );
        currentFrame++;
    }
}

// skip the chrominance bytes (we are only using luma bytes currently)
if( index < filelen )

```

```

        {
            buffer += chromsize; // adjust pointer address to next set of luminance bytes
            index += chromsize; // adjust index to next set of luminance bytes
        }
    }
    // completed!
    printf( "\rProcessing frame: %lu/%lu (100.00%%)\n", framecount, framecount );

    buffer = reset;
    compare = rstCompare;
    psnrValue = psnr( buffer, compare );
    if( psnrValue == 99.0f )
    {
        printf( "\nThe output video is exactly the same as the original video.\n\n" );
    }
    else
    {
        printf( "\nPSNR of output video: %f dB (p=%u, t=%u, lvt=%f)\n\n", psnrValue,
            plain, textured, lowVarTh );
    }

    if(outputTruncatedFile)
    {
        // open file using "w+b" = write/update binary file access mode
        fileptr = fopen( outputstr, "w+b" );

        // if fopen returns NULL pointer it failed to open the file
        if( fileptr == NULL )
        {
            printf( "Could not write output file. Exiting program.\n" );
            exit( EXIT_FAILURE );
        }
        else // file opened successful -> write back YUV data
        {
            buffer = reset; // reset buffer pointer address to write data
            printf( "Created file name: %s, now writing data...\n", outputstr );
            fwrite( buffer, 1, filelen, fileptr ); // write the data to the file
            fclose( fileptr ); // close the file
        }

        printf( "Data was successfully output!\n" );
    }

    printf( "Freeing up allocated memory\n" );
    buffer = reset; // reset buffer pointer address to free memory
    free( buffer ); // deallocate memory block
}
}

printf( "\nLow Variance (Plain) MB Count: %u / %u (%f%%)\n", (uint32_t) plainMBCount,
    totalMBCount, plainMBCount * 100.0f / totalMBCount );

printf( "\nPSNR of truncated video: %f dB\n\n", psnrValue );

if(outputCSV)
{
    printf( "Plain %% per frame was output to .csv file!\n\n" );
}

return 0; // program success
}

```



## APPENDIX D. SYNAPTIC STORAGE FOR DEEP LEARNING MODEL CODE

```
# Synaptic Storage for Deep Learning Neural Network Model Code
# Jonathon Edstrom - 2017
# Allows for adjusting 6T and 8T cells, their failure rates,
# ability to inject faults, and correct bits based on patterns
# Department: NDSU ECE Graduate Research
# Project: Data-Driven Intelligent Efficient Synaptic Storage for Deep Learning

import random
import sys, os, struct, datetime, time
import h5py
import numpy as np
from ctypes import *

#define parameters
batch_size = 128
num_epochs = 20
hidden_size = 20
failure_rate_6T = float(sys.argv[1])
failure_rate_8T = 0
seed_value = int(sys.argv[3])
correction = int(sys.argv[2])
weights_seed = seed_value + int(failure_rate_6T * 1000)
corr_bit_num = int(sys.argv[4])

np.random.seed(seed_value)

from keras.datasets import mnist # subroutines for fetching the MNIST dataset
from keras.models import Model # basic class for specifying and training a neural network
from keras.models import load_model
from keras.layers import Input, Dense # the two types of neural network layer we will be using
from keras.utils import np_utils # utilities for one-hot encoding of ground truth values
from keras.callbacks import Callback, EarlyStopping
from keras import initializers

# Convert float32 to binary (IEEE 754 single precision floating point number)
def binary(num):
    return ''.join(bin(ord(c)).replace('0b', '').rjust(8, '0') for c in struct.pack('!f', num))

# Inject Faults into a dataset of weights
def inject_faults_weights(dataset):
    if correction == 0:
        i = 0
        j = 0
        for rows in dataset:
            for cols in rows:
                for bit in range(0,30):
                    randomNum = random.random()
                    # inject failures based on failure rate
                    if randomNum <= failure_rate_6T:
                        bits = cast(pointer(c_float(dataset[i][j])), POINTER(c_int32)).contents.value
                        bits ^= (1 << bit)
                        dataset[i,j] = cast(pointer(c_int32(bits)), POINTER(c_float)).contents.value

                    j += 1
                i += 1
                j = 0

    if correction == 1:
        i = 0
        j = 0
        for rows in dataset:
            for cols in rows:
                for bit in range(0,30):
                    randomNum = random.random()
                    # inject failures based on failure rate
                    if randomNum <= failure_rate_6T:
                        bits = cast(pointer(c_float(dataset[i][j])), POINTER(c_int32)).contents.value
```

```

pre_fault_bits = bits
post_fault_bits = 0
if bit >= corr_bit_num:
    bits ^= (1 << bit)
    w1 = (bits >> 30) & 1
    if w1 == 0:
        bits |= (1 << bit)
        post_fault_bits = bits
else:
    bits ^= (1 << bit)
if post_fault_bits != 0 and post_fault_bits != pre_fault_bits:
    with open('ERRORS_failure={}_correction={}.txt'.format(failure_rate_6T,
        correction), 'a') as f:
        pre_float = cast(pointer(c_int32(pre_fault_bits)),
            POINTER(c_float)).contents.value
        post_float = cast(pointer(c_int32(post_fault_bits)),
            POINTER(c_float)).contents.value
        f.write("w{},{}, {}, {}, {}, {} \n".format(bit, pre_fault_bits, pre_float,
            post_fault_bits, post_float, post_float/pre_float))
    dataset[i,j] = cast(pointer(c_int32(bits)), POINTER(c_float)).contents.value
j += 1
i += 1
j = 0

# Inject Faults into a dataset of biases
def inject_faults_bias(dataset):
    if correction == 0:
        i = 0
        for rows in dataset:
            for bit in range(0,30):
                randomNum = random.random()

                # inject failures based on failure rate
                if randomNum <= failure_rate_6T:
                    bits = cast(pointer(c_float(dataset[i])), POINTER(c_int32)).contents.value
                    bits ^= (1 << bit)
                    dataset[i] = cast(pointer(c_int32(bits)), POINTER(c_float)).contents.value
                    i += 1

if correction == 1:
    i = 0
    for rows in dataset:
        for bit in range(0,30):
            randomNum = random.random()

            # inject failures based on failure rate
            if randomNum <= failure_rate_6T:
                bits = cast(pointer(c_float(dataset[i])), POINTER(c_int32)).contents.value
                pre_fault_bits = bits
                post_fault_bits = 0
                if bit >= corr_bit_num:
                    bits ^= (1 << bit)
                    w1 = (bits >> 30) & 1
                    if w1 == 0:
                        bits |= (1 << bit)
                        post_fault_bits = bits
                else:
                    bits ^= (1 << bit)
                if post_fault_bits != 0 and post_fault_bits != pre_fault_bits:
                    with open('ERRORS_failure={}_correction={}.txt'.format(failure_rate_6T,
                        correction), 'a') as f:
                        pre_float = cast(pointer(c_int32(pre_fault_bits)),
                            POINTER(c_float)).contents.value
                        post_float = cast(pointer(c_int32(post_fault_bits)),
                            POINTER(c_float)).contents.value
                        f.write("b{},{}, {}, {}, {}, {} \n".format(bit, pre_fault_bits, pre_float,
                            post_fault_bits, post_float, post_float/pre_float))
                    dataset[i] = cast(pointer(c_int32(bits)), POINTER(c_float)).contents.value
                    i += 1

```

```

def main():

    if corr_bit_num != 26 and corr_bit_num != 28:
        print('bad corr_bit_num!!!')
        exit()

    print('Weights seed: {}\n\n'.format(weights_seed))

    class TrainingHistory(Callback):

        def on_train_begin(self, logs={}):
            self.i = 0
            with open('ERRORS_failure={}_correction={}.txt'.format(failure_rate_6T, correction), 'a')
                as f:
                f.write("bit,pre_fault_bits,pre_float,post_fault_bits,post_fault,error\n")

        def on_epoch_begin(self, epoch, logs={}):
            with open('ERRORS_failure={}_correction={}.txt'.format(failure_rate_6T, correction), 'a')
                as f:
                f.write("\nEPOCH {} of {}:\n\n".format(epoch+1, num_epochs))

        def on_batch_end(self, batch, logs={}):
            self.i += 1

            # save the weights into a file
            model.save_weights('new_model_weights.h5')

            # access weights memory locations
            wf = h5py.File('new_model_weights.h5', 'r+')
            data_w_1 = wf['dense_1/dense_1/kernel']
            data_b_1 = wf['dense_1/dense_1/bias']
            data_w_2 = wf['dense_2/dense_2/kernel']
            data_b_2 = wf['dense_2/dense_2/bias']
            data_w_3 = wf['dense_3/dense_3/kernel']
            data_b_3 = wf['dense_3/dense_3/bias']

            # seed random number generator to keep faults in same position
            random.seed(int(seed_value))

            # inject faults into weights/biases directly
            inject_faults_weights(data_w_1)
            inject_faults_bias(data_b_1)
            inject_faults_weights(data_w_2)
            inject_faults_bias(data_b_2)
            inject_faults_weights(data_w_3)
            inject_faults_bias(data_b_3)

            # close the file containing the edited weights
            wf.close()

            # load edited weights into the model
            model.load_weights('new_model_weights.h5')

    num_train = 60000 # there are 60000 training examples in MNIST
    num_test = 10000 # there are 10000 test examples in MNIST

    height, width, depth = 28, 28, 1 # MNIST images are 28x28 and greyscale
    num_classes = 10 # there are 10 classes (1 per digit)

    (X_train, y_train), (X_test, y_test) = mnist.load_data() # fetch MNIST data

    X_train = X_train.reshape(num_train, height * width) # Flatten data to 1D
    X_test = X_test.reshape(num_test, height * width) # Flatten data to 1D
    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')
    X_train /= 255 # Normalise data to [0, 1] range
    X_test /= 255 # Normalise data to [0, 1] range

    Y_train = np_utils.to_categorical(y_train, num_classes) # One-hot encode the labels
    Y_test = np_utils.to_categorical(y_test, num_classes) # One-hot encode the labels

```

```

inp = Input(shape=(height * width,)) # Our input is a 1D vector of size 784
hidden_1 = Dense(hidden_size, activation='relu', init=initializers.random_normal(mean=0.0,
stddev=0.05, seed=weights_seed))(inp) # First hidden ReLU layer
hidden_2 = Dense(hidden_size, activation='relu', init=initializers.random_normal(mean=0.0,
stddev=0.05, seed=weights_seed))(hidden_1) # Second hidden ReLU layer
out = Dense(num_classes, activation='softmax', init=initializers.random_normal(mean=0.0,
stddev=0.05, seed=weights_seed))(hidden_2) # Output softmax layer

model = Model(input=inp, output=out) # Define the model by specifying input and output layers

model.compile(loss='categorical_crossentropy', # using the cross-entropy loss function
optimizer='adagrad', # using the Adagrad optimiser
metrics=['accuracy']) # reporting the accuracy

history = TrainingHistory()

hist = model.fit(X_train, Y_train, # Train the model using the training set...
batch_size=batch_size, nb_epoch=num_epochs,
verbose=1, validation_split=0.1, callbacks=[history])

with open('output_failure={}_correction={}.txt'.format(failure_rate_6T, correction), 'a') as f:
    f.write("acc:\n")
    f.write('\n'.join(str(element) for element in hist.history["acc"]))
    f.write("\n\nval_acc:\n")
    f.write('\n'.join(str(element) for element in hist.history["val_acc"]))
    f.write("\n\nloss:\n")
    f.write('\n'.join(str(element) for element in hist.history["loss"]))
    f.write("\n\nval_loss:\n")
    f.write('\n'.join(str(element) for element in hist.history["val_loss"]))

scores = model.evaluate(X_test, Y_test, verbose=0) # Evaluate trained model on the test set
with open('output_failure={}_correction={}.txt'.format(failure_rate_6T, correction), 'a') as f:
    f.write("\n\nAccuracy: %.2f%%\n" % (scores[1]*100))
print("\nAccuracy: %.2f%%" % (scores[1]*100))

# if running directly from command line
if __name__ == "__main__":
    if (True):
        main()
    else:
        print('Incorrect number of arguments... Exiting.')

```

## APPENDIX E. MNIST BIT FAULT INJECTION PROGRAM

```
/*
  MNIST Bit Fault Injection Program
  Jonathon Edstrom - 2018-2019
  Injects faults to MNIST Training/Test Dataset
  Department: NDSU ECE Graduate Research
*/

// includes
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <math.h>
// Mersenne Twister Random Number Generator
#include "twister.h" // https://www.mcs.anl.gov/~kazutomo/hugepage-old/twister.c

typedef enum { false, true } bool;

// instantiate globals
FILE *fileptr; // pointer for YUV data input/output files
unsigned char *buffer; // pointer for YUV data allocated memory
unsigned char *reset; // pointer value to reset the buffer to the beginning
unsigned char temp; // holds the byte value while faults are injected
unsigned long long int filelen; // length of the input file (total bytes)
// decimal that sets how often a bit has a fault
long double s7_error, s6_error, s5_error, s4_error, s3_error, s2_error, s1_error, s0_error;
// for counting the amount of errors we actually apply for each bit
long int s7_errorCount, s6_errorCount, s5_errorCount, s4_errorCount,
        s3_errorCount, s2_errorCount, s1_errorCount, s0_errorCount;
long double randomNumber; // used for storing random numbers for inputting faults
unsigned int seedValue; // seed value for random number generator
unsigned int sramWidth, sramHeight; // dimensions of the SRAM in terms of bit-cells
unsigned long long int errorCounter = 0, totalCounter = 0; // verify error rate works
double s7_errorPercentage, s6_errorPercentage, s5_errorPercentage, s4_errorPercentage,
        s3_errorPercentage, s2_errorPercentage, s1_errorPercentage, s0_errorPercentage;

// function definitions
unsigned char applyFaults( unsigned char );
unsigned char injectFault( unsigned char, long double );

// application entry point
int main(int argc, char * argv[])
{
    printf( "\nMNIST Training Data Bit Fault Injection Program (J.E. 2018-2019)\n" );

    // seed the Mersenne Twister PRNG
    seedMT((unsigned) time(NULL));

    if( argc != 14 && argc != 15 ) // argc should be 13 or 14 for correct execution
    {
        // print argv[0] assuming it is the program name with the following usage hint to user
        printf ( "usage: %s filename sramWidth sramHeight outputFile? s7_error s6_error s5_error
                s4_error s3_error s2_error s1_error s0_error unique_id [optional]seedValue\n",
                argv[0] );
    }
    else // correct number of arguments
    {
        // initialize global variables
        sramWidth = atoi( argv[2] ); // test chip from paper uses 256
        sramHeight = atoi( argv[3] ); // test chip from paper uses 256
        s7_error = atof( argv[5] );
        s6_error = atof( argv[6] );
        s5_error = atof( argv[7] );
        s4_error = atof( argv[8] );
        s3_error = atof( argv[9] );
        s2_error = atof( argv[10] );
        s1_error = atof( argv[11] );
    }
}
```

```

s0_error = atof( argv[12] );
    unsigned int unique_id = atoi( argv[13] );

    printf( "Unique ID: %u\n", unique_id );

// set up random number generator
if( argc == 15 )
{
    seedValue = atoi( argv[14] );
    srand(seedValue); // seed to user specified argument value
}
else
{
    seedValue = (unsigned) time(NULL);
    srand( seedValue ); // seed to random number using time
}

// assume argv[1] is the filename to open
// open file using "rb" = read binary file access mode
fileptr = fopen( argv[1], "rb" );

// if fopen returns a NULL pointer it failed to open the file
if( fileptr == NULL )
{
    printf( "Could not open file: %s. Exiting program.\n", argv[1] );
    exit( EXIT_FAILURE );
}
else // files opened successfully -> allocate memory buffer space
{
    printf( "Input file opened successfully!\n" );
    fseek( fileptr, 0, SEEK_END ); // jump to end of file
    filelen = ftell( fileptr ); // get current byte offset in file compare file

    //printf( "Size of input file in bytes: %llu\n", filelen );

    // verify that the file is the correct size
    if( filelen != 47040016 && filelen != 7840016 )
    {
        printf( "Input file should be 47,040,016 bytes (training images) or 7,840,016 bytes (test
            images) in size. Please check the input file. Exiting program.\n" );
        exit( EXIT_FAILURE );
    }

    rewind( fileptr ); // jump to beginning of file

    buffer = ( unsigned char * ) malloc( filelen + 1 ); // enough memory for file + \0 (EOF)
    reset = buffer;
    if( buffer == NULL )
    {
        printf( "Failed to allocate memory for file. Exiting program.\n" );
        exit( EXIT_FAILURE );
    }
    else
    {
        printf( "Memory allocated successfully!\n" );
    }

    fread( buffer, 1, filelen, fileptr ); // read file into memory

    fclose( fileptr ); // close the file

    buffer = reset; // reset buffer pointer address to beginning
    buffer += 16; // skip first 16 bytes (header info = not part of the training/test dataset)

    unsigned long long int dataset_index = 16; // current array index (skip 16 byte header)
    unsigned long long int sram_index = 0; // keeps track of SRAM byte index
    while( dataset_index < filelen ) // loop until EOF
    {
        srand(seedValue); // reset location of faults for each frame
        // loop until we hit the maximum size the 'SRAM prototype' can hold
        while( sram_index < (sramWidth * sramHeight / 8) && dataset_index < filelen )

```

```

    {
        temp = *buffer; // store current byte
        *buffer = applyFaults( temp ); // truncate

        // increment buffer address, dataset_index and sram_index variables
        buffer++;
        dataset_index++;
        sram_index++;
    }

    sram_index = 0; // reset SRAM byte index

    // calculate progress and print
    //printf( "\rProgress: %f%%", dataset_index * 100.0f / filelen );
}

//printf( "\n" );

// calculate the actual error percentages
s7_errorPercentage = (float)s7_errorCount / ( (float)totalCounter / 8 );
s6_errorPercentage = (float)s6_errorCount / ( (float)totalCounter / 8 );
s5_errorPercentage = (float)s5_errorCount / ( (float)totalCounter / 8 );
s4_errorPercentage = (float)s4_errorCount / ( (float)totalCounter / 8 );
s3_errorPercentage = (float)s3_errorCount / ( (float)totalCounter / 8 );
s2_errorPercentage = (float)s2_errorCount / ( (float)totalCounter / 8 );
s1_errorPercentage = (float)s1_errorCount / ( (float)totalCounter / 8 );
s0_errorPercentage = (float)s0_errorCount / ( (float)totalCounter / 8 );

// create output file name string
int len = strlen( argv[1] ); // get length of input file name
char filename[len];
strcpy( filename, argv[1] ); // get input file name
len = len + 100; // add correct amount for output file naming
char outputstr[len];
strcpy( outputstr, filename );
strcat( outputstr, "_faults_" );
    strcat( outputstr, argv[13] );

    //printf( "Output filename: %s\n", outputstr );

// output to file
if( atoi( argv[4] ) == 1 )
{
    // open file using "w+b" = write/update binary file access mode
    fileptr = fopen( outputstr, "w+b" );

    // if fopen returns NULL pointer it failed to open the file
    if( fileptr == NULL )
    {
        printf( "Could not write output file. Exiting program.\n" );
        exit( EXIT_FAILURE );
    }
    else // file opened successful -> write back YUV data
    {
        buffer = reset; // reset buffer pointer address to write data
        //printf( "Created file: %s, now writing data...\n", outputstr );
        fwrite( buffer, 1, filelen, fileptr ); // write the data to the file
        fclose( fileptr ); // close the file
    }
}

//printf( "Freeing up allocated memory...\n" );
buffer = reset; // reset buffer pointer address to free memory
free( buffer ); // deallocate memory block

if( atoi( argv[4] ) == 1 )
{
    printf( "Data was successfully output!\n" );
}

printf( "S7 Error Rate: %.10f\n", s7_errorPercentage );

```

```

    printf( "S6 Error Rate: %.10f\n", s6_errorPercentage );
    printf( "S5 Error Rate: %.10f\n", s5_errorPercentage );
    printf( "S4 Error Rate: %.10f\n", s4_errorPercentage );
    printf( "S3 Error Rate: %.10f\n", s3_errorPercentage );
    printf( "S2 Error Rate: %.10f\n", s2_errorPercentage );
    printf( "S1 Error Rate: %.10f\n", s1_errorPercentage );
    printf( "S0 Error Rate: %.10f\n", s0_errorPercentage );
}
}

return 0; // program success
}

// helper function to apply faults to MNIST data byte
unsigned char applyFaults( unsigned char byte )
{
    // instantiate local variables
    unsigned int bit_position;
    unsigned char value, bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8, newbit1, newbit2, newbit3,
        newbit4, newbit5, newbit6, newbit7, newbit8, output;

    // get individual bits from byte
    for( bit_position = 0; bit_position < 8; bit_position++)
    {
        switch( bit_position ) {
            case 0:
                value = ( byte & 128 ); // bitwise AND with 10000000
                if( value > 0 )
                {
                    bit1 = 1;
                }
                else
                {
                    bit1 = 0;
                }
                newbit1 = injectFault( bit1, s7_error );
                if( bit1 != newbit1 )
                {
                    s7_errorCount++;
                }
                break;
            case 1:
                value = ( byte & 64 ); // bitwise AND with 01000000
                if( value > 0 )
                {
                    bit2 = 1;
                }
                else
                {
                    bit2 = 0;
                }
                newbit2 = injectFault( bit2, s6_error );
                if( bit2 != newbit2 )
                {
                    s6_errorCount++;
                }
                break;
            case 2:
                value = ( byte & 32 ); // bitwise AND with 00100000
                if( value > 0 )
                {
                    bit3 = 1;
                }
                else
                {
                    bit3 = 0;
                }
                newbit3 = injectFault( bit3, s5_error );
                if( bit3 != newbit3 )
                {
                    s5_errorCount++;
                }
            }
        }
    }
}

```



```

    }
    break;
case 3:
    value = ( byte & 16 ); // bitwise AND with 00010000
    if( value > 0 )
    {
        bit4 = 1;
    }
    else
    {
        bit4 = 0;
    }
    newbit4 = injectFault( bit4, s4_error );
    if( bit4 != newbit4 )
    {
        s4_errorCount++;
    }
    break;
case 4:
    value = ( byte & 8 ); // bitwise AND with 00001000
    if( value > 0 )
    {
        bit5 = 1;
    }
    else
    {
        bit5 = 0;
    }
    newbit5 = injectFault( bit5, s3_error );
    if( bit5 != newbit5 )
    {
        s3_errorCount++;
    }
    break;
case 5:
    value = ( byte & 4 ); // bitwise AND with 00000100
    if( value > 0 )
    {
        bit6 = 1;
    }
    else
    {
        bit6 = 0;
    }
    newbit6 = injectFault( bit6, s2_error );
    if( bit6 != newbit6 )
    {
        s2_errorCount++;
    }
    break;
case 6:
    value = ( byte & 2 ); // bitwise AND with 00000010
    if( value > 0 )
    {
        bit7 = 1;
    }
    else
    {
        bit7 = 0;
    }
    newbit7 = injectFault( bit7, s1_error );
    if( bit7 != newbit7 )
    {
        s1_errorCount++;
    }
    break;
case 7:
    value = ( byte & 1 ); // bitwise AND with 00000001
    if( value > 0 )
    {
        bit8 = 1;
    }

```

```

    }
    else
    {
        bit8 = 0;
    }
    newbit8 = injectFault( bit8, s0_error );
    if( bit8 != newbit8 )
    {
        s0_errorCount++;
    }
    break;
default:
    printf( "Something went wrong. Exiting program.\n" );
    exit( EXIT_FAILURE );
}
}

// construct byte from bits after faults were applied
output = ( ( newbit1 << 7 ) | ( newbit2 << 6 ) | ( newbit3 << 5 ) | ( newbit4 << 4 ) |
           ( newbit5 << 3 ) | ( newbit6 << 2 ) | ( newbit7 << 1 ) | newbit8 );

return output;
}

// function that will inject a fault to the input bit according to the error rate
unsigned char injectFault( unsigned char bit, long double errorRate )
{
    bool fault = false;

    // Don't divide by 0
    if( errorRate != 0.0 )
    {
        // get a random number based on the Mersenne Twister Algorithm
        unsigned long twisterRandomNumber = randomMT();
        long double randomFloat = (double)twisterRandomNumber/4294967295;
        if(randomFloat <= errorRate)
        {
            fault = true;
        }
    }

    if( fault )
    {
        totalCounter++;
        // apply fault
        if( bit == 0 )
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
    else
    {
        // no fault
        totalCounter++;
        return bit;
    }
}
}

```