# EVALUATION OF ML INFERENCE WORKLOADS ON REDUCED REFRESH RATE DRAM

**Daniyar Zhakiyev,** B.Eng. in Electrical and Computer Engineering

**Submitted in fulfilment of the requirements**

**for the degree of Master of Science**

**in Electrical and Computer Engineering**

NAZARBAYEV UNIVERSITY

**School of Engineering and Digital Sciences**

**Department of Electrical and Computer Engineering**

53 Kabanbay Batyr Avenue,

Nur-Sultan, Kazakhstan, 010000

**Supervisors**: Nursultan Kabylkas, Daniele Tosi

**01.05.2024**

DECLARATION

I hereby, declare that this manuscript, entitled “Evaluation Of Ml Inference Workloads on Reduced Refresh Rate Dram”, is the result of my own work except for quotations and
citations which have been duly acknowledged.
I also declare that, to the best of my knowledge and belief, it has not been previously or concurrently submitted, in whole or in part, for any other degree or diploma at Nazarbayev University or any other national or international institution.

[illegible]
-------------------------------------------
Name: Daniyar Zhakiyev
Date: 05.04.2024

## Abstract

This thesis investigates the potential benefits of reducing DRAM refresh rates to improve the performance of Machine learning and Neural Network inference workloads. The increasing integration of ML and NN models in various industries makes it necessary to optimize these models to efficiently use computing resources, particularly in devices with limited capabilities. To maintain data integrity, DRAM requires periodic refresh cycles, which has a substantial impact on power consumption and system efficiency. Thus, DRAM refresh rates can be lowered for performance purposes. While this study does not add any additional components to the memory controller, other proposed approaches had hardware or software overhead. Preliminary findings indicate that NNs have a remarkable tolerance to data loss, caused by reduced refresh rates. Results show that DRAM refresh rates can be reduced by up to 15-150 times the usual refresh rate without significant impact on NN accuracy. Additionally, NNs showed 2.7% faster inference and consumed 5.6% less power at refresh rate of 1 second.

# Table of Contents

# Chapter 1 – Introduction

Neural Networks (NN) and Machine Learning (ML) are one of the leading technological breakthroughs that emerged as an irreplaceable tool across a wide range of industries, including healthcare, finance, and automotive industry. For real-time AI applications, it is becoming more and more important to integrate ML and NN models onto a variety of devices, ranging from mobile gadgets to a high-end server [1]. However, deploying such computationally complex models on devices with limited resources presents considerable problems, especially in terms of processing power and energy efficiency.

Despite the recent advances in ML field that resulted in more efficient algorithms and compact models, these solutions still require significant processing resources. The challenge is particularly noticeable in mobile devices, which are limited by processing power and energy consumption even if they are widely available [2]. Addressing these limitations is critical for achieving the full potential of on-device AI applications, demanding novel optimization strategies that can enhance speed while preserving model accuracy and user experience.

One potential area for optimization is Dynamic Random Access Memory (DRAM), an essential element of computing systems that stores data and program instructions for processors. DRAM is noted for its high density at low cost, making it an ideal choice for quick memory storage in a wide range of devices. However, DRAM's reliance on tiny capacitors to store data causes charge leakage over time, necessitating periodic refresh operations to ensure data integrity [3]. These refresh procedures consume a large fraction of the system's power, affecting overall performance, particularly in systems that require high efficiency.

Some researches suggests that DRAM refresh rates can be significantly reduced to improve both performance and power consumption [4]. By adjusting the DRAM refresh rate, it is possible to reduce the overhead associated with refresh operations while retaining the accuracy and reliability of NN inferences. This method has the potential to increase the deployment of advanced NN models on a wider range of devices, especially those in which power efficiency and computing constraints are particularly hard. NNs are more suitable for this approach compared to ML algorithms due to their distributed architecture with nodes and weights that make the network more robust.

The objective of this research is to explore the potential for improving Neural Network performance through the reduction of the DRAM refresh rate, examining how decreased refresh intervals might impact the accuracy and power efficiency of NNs during inference. The hypothesis is that Neural Networks are inherently error-tolerant to some extent, thus allowing them to manage small data loss from reducing the DRAM refresh rate.

# Chapter 2 - Background

## 2.1. DRAM Overview

### 2.1.1 DRAM organization

Dynamic Random Access Memory (DRAM) is an essential component of modern computing systems, acting as the primary volatile memory used by the CPU (Central Processing Unit) to store data during operation. DRAM offers significantly faster access thus its architecture strikes an optimal balance between density, speed, power consumption, and cost. DRAM has an internal hierarchy of *ranks*, *chips* and *banks* at the high-level organization. Within a DRAM rank, there are several chips as in Figure 1 that further consist of banks. Banks have rows and columns of DRAM cells that are organized in a 2D array. The typical length of one DRAM row is 8192 cells.

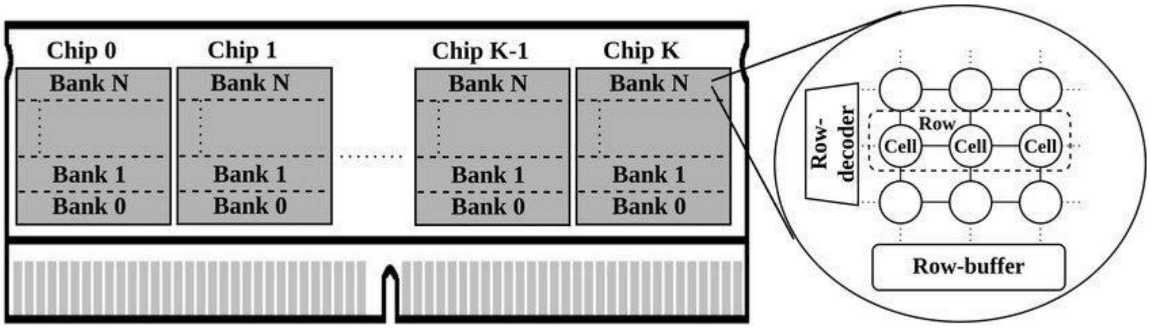
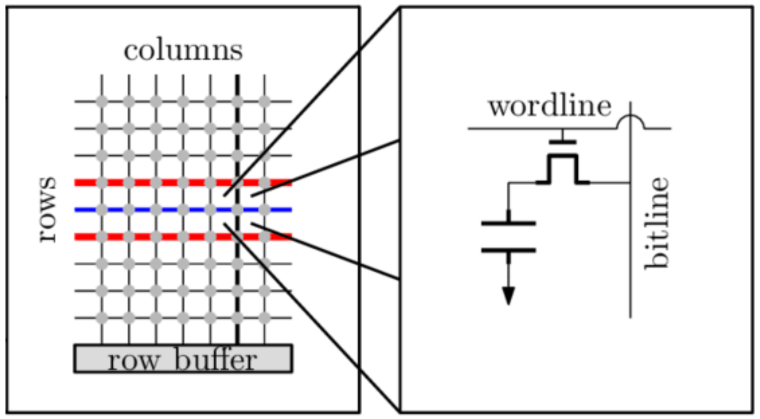


**Figure 1. DRAM Rank organization** [5]

From Figure 2b single DRAM cell is a simple circuit constructed from a *capacitor* and *transistor*. Capacitor is the main element in a cell, it stores the charge which symbolizes a single bit of binary information: charged capacitor represents 1, and discharged represents 0. The transistor operates as a switch, allowing the capacitor to be charged or discharged. Each DRAM cell is connected to the *bitline* and *wordline*. A bitline, which is responsible for binary data transport, connects cells vertically to columns. Horizontally, cells are connected to a row via a worldline, which activates the transistor; when a worldline is triggered, the bitline can access the cell.

Bitlines attach to *row buffers* or *sense amplifiers*, which can detect cell charge and then buffer binary data.



a. Rows in a single bank b. Single cell

**Figure 2. DRAM Bank structure** [6]

### 2.1.2 DRAM Access

To read/write DRAM bank 3 stages are involved:

*Initial Stage.* During the initial stage, all cells are precharged and can be activated

*Row Activation & Sense Amplification*. The worldline is triggered by an *ACT* (activation) command, thus connecting the cells to bitline, so that bitline voltage transfers to the cells or out of the cells depending on the initial charge in the cell's capacitor. The change of voltage in the bitline is sensed by a sense amplifier that subsequently amplifies the signal and latches it in a row buffer in a binary format. After this row buffer is ready to be accessed. In DDR4 [7] standards, time spent on this stage is called *activation latency* and is defined as *tRCD*.

*Read/Write.* Data in the row buffer can be accessed by *READ/WRITE* commands, with column access times denoted as *tCL* for reads and *tCWL* for writes. After or during this stage cells are restored to their initial state, it is called *restoration latency tRAS*.

*Precharge.* In the last stage, the *precharge command* (*PRE*) is used to access the different rows. This command deactivates the worldline, and thus disconnects the cell from bitline. Bitline voltage is normalized and is ready to sense different row voltages. Time spent on this stage is called *precharge latency tRP* [8].

### 2.1.3 DRAM Refresh

The most important aspect of DRAM for this work is DRAM refresh. In Figure 2b DRAM cell is depicted to have a capacitor. In reality, the capacitors in DRAM cells are manufactured to be very small, the average capacitance of DRAM cell is 25 fF [9]. However, this leads to the drawback of tiny capacitors called leakage current. Due to the natural properties of small capacitors the charge stored in the capacitor tends to leak over time, so the data stored in DRAM cell can be lost.

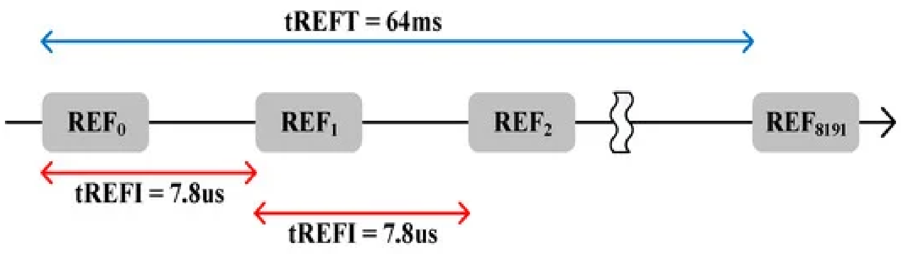


**Figure 3. Refresh timings in DRAM** [10]

To tackle this problem DRAM cells need to be periodically refreshed. The minimum time at which a cell can hold the charge in the capacitor is described as *retention time.* Thus refresh operation has to be performed on each row every retention time *tRET or tREFW*. However due to the large number of rows in each bank refresh commands are distributed throughout all rows periodically during retention time. Refreshes are performed via the Auto-Refresh (AREF) command. The retention time parameter is declared for every DRAM by the Joint Electron Device Engineering Council (JEDEC) which develops unified standards for the microelectronics industry [11]. According to JEDEC [7] standards minimal retention time of a DRAM cell is **64ms** at temperatures <85°C and **32ms** at >85°C. There are 8192 row-groups in a bank, thus there are 8192 AREF commands that need to be performed within 32ms. This refresh period is called *tREFI* and is calculated by (2.1).

$$tREFI = \frac{64\,ms}{8192} = 7.8\,\mu s \quad (< 85°C) \tag{2.1}$$

$$tREFI = \frac{64\,ms}{8192} = 3.9\,\mu s \quad (> 85°C) \tag{2.2}$$

Some of the DRAM timings proposed in this section and additional useful timings can be summarized:

**Table 1. DRAM timings**

| Timing | Description |
|---|---|
| tREFW | Period to refresh every row in a DRAM bank. |
| tCKE | Stabilization time for the clock before actions. |
| tFAW | Allowed activation period for rows across four banks. |
| tPRDI | Reactivation time from low power mode. |
| tRAS | Duration a row stays active before precharging. |
| tRCD | Delay from row activation to read/write. |
| tREFI | Average interval between refreshes. |
| tRFC | Time to finish a row's refresh operation. |
| tRP | Time to deactivate a row before accessing another. |
| tRRD | Minimum gap between activating different rows. |
| tRTP | Interval from reading to row precharging. |
| tWTR | Wait time from a write to the next read. |
| tZQI | Frequency of ZQ calibration for impedance |

## 2.2. Neural Networks

Neural networks (NN) are a collection of algorithms used to detect patterns in data that are presented to them. They are modeled after the human brain's neurons and neural connections to mimic the brain's ability to adapt to new information. Neural networks are built from neurons that are interconnected with each other in a network-like structure as seen in Figure 4. In a software representation of NN neurons are called *nodes* and the links between nodes are called *weights.*

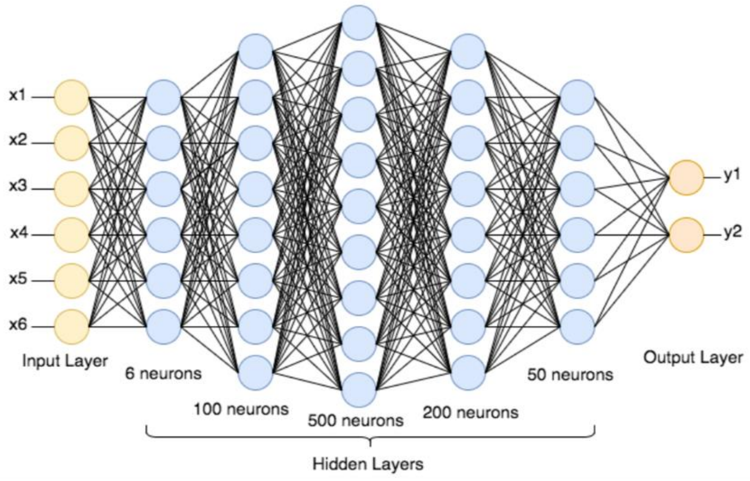


**Figure 4. Basic Neural Network structure** [12]

The basic structure of the Neural Network can be observed in Figure 4. It includes 3 main layers [12]: i) *Input Layer* where the data is fed to the system to be categorized and passed to the next layers. ii) *Hidden Layers* process the information from the input layer with mathematical functions and pass it to other layers within the hidden layer. There can be many hidden layers and millions of nodes in them. iii) *Output Layer* provides the final result of computations. The output layer can have one or multiple nodes depending on the task.

There several types of Neural Networks that are suitable for different tasks due to their unique structures. Feedforward Neural Networks (FNNs) process the data through one forward way and use feedback to improve through iterations [13]. Convolutional Neural Networks (CNNs) use convolution functions to process data, and thus are ideal at grid-based data processing, such as image recognition and medical image recognition [14]. Recurrent Neural Networks (RNNs) excel at sequential data analysis like speech and text recognition, and used widely in natural language processing (NLP) .

Training stage is done to teach the Neural Network to perform a task specifically in a way the user needs it. During the training phase, NN is exposed to a large amount of labeled or unlabeled data to identify patterns and make predictions of the output. This stage is called forward pass. Next, a backward pass or backpropagation creates the feedback loop to improve the predictions by updating weights.

After training Neural Networks can be used for *inference.* Inference is the process when trained NN is utilized to make predictions on unseen data. This stage is less computationally intensive compared to training; it uses the weights generated during training. However, NN inference is still resource resource-heavy operation that uses a lot of computational power, especially for large models. The major difficulty in inference is to keep the balance between accuracy and efficiency.

# Chapter 3 – Literature review

There is a substantial body of research regarding the reduced refresh rate DRAM field. Some works cover the DRAM latency mitigation techniques, and some focus on special low latency architectures. In the core all studies are connected to the DRAM timings and most importantly tREFW or retention time. The retention time set in JEDEC [7], [8], [11] standards is 64ms, all cells are refreshed with 64ms periodicity, however according to Liu et al. [4] only 0.00000001% of cells from $10^{11}$cells have retention time less than 256ms. Leaky cells or weak cells are the terms used to describe cells that have low retention time.

## 3.1. Low Latency DRAM

Regarding the mitigation of DRAM latency several methods are proposed and can be divided to 3 main categories: *i) Retention aware ii) Row-level and iii) Refresh schedules.*

### 3.1.1. Retention aware

An innovative method for DRAM memory optimization is presented by [4]. RAIDR uses mechanisms to lower the refresh energy of DRAM by intelligently adjusting each cell's retention characteristics to determine the refresh rate. In this approach DRAM rows are separated into bins where cells are profiled and classified according to their retention time. RAIDR lowers the refresh frequency for bins with high retention time, which account for the vast majority of cells and saves more frequent refreshes for bins with leaky rows. RAIDR shows energy used for DRAM refresh can be significantly lowered, which would improve system performance overall, especially on multi-core systems as DRAM capacity keep rising. This approach shows refresh rate reduction by 74.6% for only a 0.3% loss in data integrity, however it comes with storage overhead of 1.25 KB in memory controller .

In RAPID [15] the variance of retention time of DRAM cells through the rank is utilized. This method prioritizes the the pages with longer retention time, so that by using only them a single longer refresh period could be used. In this approach no hardware overhead is present, because it is a software-based solution. RAPID manages to reach 83% - 95%  in refresh energy saved for various setups.

#### 3.1.2. Row level

ESKIMO [16] uses the information about used and freed space from memory controller. During the time when specific memory region is not used this approach do not refresh the unused memory region. By performing this task average of 39% of energy is saved.

[17] proposes the technique that simultaneously prolongates the refresh latency and targets rows with weak cells with periodic read commands. By constant read requests weak cells data in them is preserved. The result of such technique is in 66% improvement of refresh enery usage and 31.8% of DRAM energy consumption. To setup this solution no hardware modifications are needed, only an additional weak row buffer in the memory controller, resulting in a highly efficient and practical solution with low overhead.

#### 3.1.3. Refresh schedules

A framework called Elastic Refresh [18] aims to reschedule the refresh commands by still following the JEDEC standards. During 1 refresh cycle of 64ms up to 9 x tREFI refresh commands can be postponed and performed at the end of the cycle. Authors of elastic refresh utilize this opportunity to hold the refreshes during the access or use of DRAM, and then quickly burst the refresh operations when memory is no longer active, so that refresh operations do not clash with refresh operations. This method shows 10-41% of performance improvement depending on workloads.

AL-DRAM [19] is the low latency architecture  that has the adaptive latency function. This method dynamically modifies the settings based on the history of cell accesses. This adaptivity allows it to boost performance and improve energy efficiency. Since the retention time in DRAM cell is not uniform AL-DRAM can make more flexible memory operations.

The stability of DRAM cells are studied under various voltages in [20]. Study is discovering a untrivial link between supply voltage, access patterns and cell failure rates. According to Khan et al. lower voltages can cause more cell failures, but some access patterns can reduce them. Most importantly this study forces the scientific community in DRAM field to show more attention to DRAM voltage levels and access patterns for DRAM dependencies.

VAMPIRE (Validating and Modeling Power Issues in Real Environments) [21] conduct an experiment on DRAM power models, showing the differences between existing models and real-world observations. Study provides a more accurate and precise power model for DRAM, which is useful for simulation credibility.

Summarizing the overview of different approaches that change the refresh latency, there are many unique and effective ways of optimizing the DRAM refresh rate to increase performance. However, all these approaches and developed techniques have a common drawback. All of them use additional software/hardware/memory controller modifications that create hardware or software overhead. This means that most of the suggested solutions will not be implemented in real DRAM device, because DRAM manufacturers squeeze the maximum capacity possible in all dram ranks, and they cannot afford to add overhead on DRAM circuit or in memory controller. Software solutions are easier in implementing; however, they still add on top of the memory controller software. Contrary the approach chosen for this thesis does not need any software or hardware add-ons, simply changing the timing parameters is all needed to do. The uselessness of more sophisticated algorithms is explained by the high error tolerance of Neural Networks.

## Chapter 3.2 NN error tolerance

This section is devoted to the error tolerance of Neural Networks. Substantial amount of literature is provided looking into the limits of error that can be induced on NN, also provides methodologies that improve error tolerance of NN.

Scientists explore means of upgrading ANN for classifying challenges to make them more tolerant of failures [22]. Both carry out simulating seeding networks and fault tolerance algorithms along with training algorithms. The experiment demonstrates that supplying some excess resources and changing the way the training algorithms are implemented can really aid in PFT. However, surprisingly, brute force method obtains better record in high PFT implementation when compared to fault-tolerant gradient descent technique. The research emphasizes 3 main areas for optimizing fault tolerance: network size, first PFT, and the number of times the PFT is repeated. Moreover, the study reveals the effects of network node congestion on the distribution of incoming data.

The paper [23] argues that by resorting to resistive RAM-Based binarized neural networks (BNNs) for energy constrained embedded artificial intelligence and studying their tenacious bit error tolerance, exciting advances can be made. A study found that BNNs were able to withstand synaptic weights in the order of up to 10^-4 bit errors, and with the use of certain training protocols, up to 4 × 10^-2 were successfully handled when they were implemented on circuit hardware that had errors (MNIST and CIFAR10). This exceptional neuroplasticity must have profound implications for memory devices' development and the optimization of the course of the devices' running on neural networks. RRAM cell variability and bit error rates could endure considerable increment in device scaling employing programming energy efficiency, achieving the area-error rate product superior by the factor of thirty turned out to be possible. BNNs are recognized as one of the powerful computing schemes that are capable of handling hardware constraints. Due to their robustness and adaptability, they can be employed in AI-related embedded applications that are deep energy efficient.

The ramifications of achieving bit error tolerance in NNs due to the use of aggressive parameter tuning which leads to bit errors is explored in [24]. The authors stress that the approach used at present to get bit error tolerance by training bit flip is very expensive, and it is also difficult to scale. They state their need for different procedures leading to bit error tolerance and, on the other hand, describe the lack of knowledge of principles of an NN bit error tolerance. The paper investigates two metrics for understanding the internal changes caused by bit flip training in binarized neural networks (BNNs): the neuron-level bit error tolerance metric and the inter-neuron bit error tolerance metric. The study proves the theoretical result that the accuracy over BER, the neuron-level bit error tolerance is closely related to BNNs. Besides that, the work introduces novel neuron importance metric and explores its role in understanding how neurons work together in making BNN more error tolerant. The results imply that BNNs of different sizes can be accounted for by the proposed metrics providing a way for inventing new techniques for achieving the bit error tolerance in NN and thus saving energy.

[25] is focused on improving power efficiency of numerical linear algebra procedures (i.e., implement fault-tolerant matrix decompositions on heterogeneous platforms with GPUs, and optimize matrix-matrix multiplication on GPUs). It also investigates iterative methods fault tolerance, deep neural networks compression methods and machine learning fault tolerance. The authors present the novel algorithm-based fault-tolerance (ABFT) schemes to save convolutional neural networks (CNN) inference processes against soft errors, proposing FT-CNN, a fault-tolerant framework for CNNs with low overhead. We conduct our experiments on ImageNet with CNN models including AlexNet, VGG-19, ResNet-18, and YOLOv2 which are popular. Experimental results show that our implementation can cope with soft errors with about 4%-8% additional runtime in both error-free and error-injected scenarios.

## Chapter 3.3 NN and DRAM

There are some approaches trying to optimize Neural Network performance with different types, methods, and modifications of DRAM. [26] describes the MViD architecture of Recurrent Neural Networks (RNNs), which significantly speeds up the Recurrent Neural Networks (RNNs) computations by proceeding them inside a computationally complex DRAM. MViD exploits data sparse matrices formation, quantization, and MAC units (multiply-add) within the DRAM banks enabling simplified computational operations Results prove that the accelerated systems engaged record a heightened throughput performance compared to the baseline systems, thus leading to increase in both capacity and energy efficacy gains with the sparsity level in the matrices. The architecture is able to do request processing efficiently from processors in MV-banks, by using commands like p-PRE, s-PRE and r-PRE compared to regular PRE command. The suggested architecture mitigates the power constraints, space usage and energy requirements, thereby giving a possible solution for improving performance and energy efficiency.

Some papers touch upon the topic of the energy accuracy tradeoffs in smart camera systems, approximate computing techniques proposed for deep learning neural networks, error propagation in deep learning applications, and the use of memory system optimizations for deep learning applications. It plans an adventurous ZEM [27] unit for refresh less DRAM in deep learning, which lessens error rates, improves performance, and enhances energy efficiency. Van ZEM via the usage of zero-error approach encoding scheme provides not the risk of the error introduction that in result cannot be acceptable. The experimental results show that ZEM can provide very good accuracy as well as successful power consumption when compared existing works so it can be applicable to many deep learning applications and indicates its potential for future DRAM generation.

One of the difficulties DRAM systems have with is refresh operations that degrade performance as DRAM chip density increases which is explained in the article. The book proposes HiRA [28] Memory Controller (HiRA-MC) and Hidden Row Activation (HiRA) which would help to mitigate or eliminate this issue. The DRAM chips remain unaltered while the refresh operation delay is hidden by the performing the operation alongside the accessing or refreshing of a different row in the same bank. According to the achievements of the experiment, the HiRA method reduces latency per 51,4%. The next generation (HiRA-MC) of the DRAM chips can boost performance, as compared to its counterparts, by 12.6%, and as much as 3.73 times in preventing Row Hammer induced failures.

[29] article explains the problem in teaching deep neural networks (DNNs) with the help of metal oxide resistive random access memory (RRAM) because of their inability to advance the

backpropagation (BP) or weight updating operations. The paper proposes a Times in Memory training architecture (TIME) based on RRAM (Resistive RAM) for its implementation as well as peripheral circuits design. Due to the adoption of TIME technique, NN training on RRAM is no longer a problem and the utilization of peripheral circuit re-usage is also optimized while nonideal factors that restrict the implementation of RRAM is addressed using optimization strategies. Computational data revealed the much higher existing energy efficiency than the conventional ones and can be enhanced if tuning costs is reduced.

The literature explains a technique of cutting power consumption in DNN systems through period extension using ECC for DRAM refresh. VECC [30] technique, which is a combination of voting mechanisms and ECC, detects and corrects any weight data errors that could occur during the retention process. Comparison to traditional methods indicates VECC to make the saving of 93% refresh power with 0.5% accuracy loss and 0.5% check bit overhead for AlexNet, ResNet, and VGG19 CNNs models trained on ImageNet. Moreover, RRAM tuning technologies can be significantly reduced.

How DRAM refresh cycles create a bandwidth bottleneck in the memory read intensive tasks is investigated in this work. It introduces DR RAM [31] that improves read throughput by permitting overlap of the read and refresh operations. DR RAM mainly covers device renewal and data recovery, with the latter also allowing data recovery during the refresh. Under just simulated situations, DR nearly eliminates the overhead refresh, resulting in up to 12% extra read bandwidth and 50-60% latency improvement for NN than current DRR4 devices.

ADROIT (Adaptive Dynamic Refresh Optimization Toolkit) [32] is an open dynamic framework that offers dynamic refresh optimization for general purpose DNNs and a variety of processing platforms. During the ADROIT training process, the refresh rate is dynamically adjusted at runtime by evaluating the loss feedback from the data, which considers data idle time, lifetime, and size, to reduce the amount of refresh operations. Simulation results demonstrate that training for DNN can save up to 98.9% on refresh energy and 24.7% on the entire DRAM energy while holding accuracy constant. ADROIT lets to automatize applications for DNNs, supported by different hardware platforms and without any manual configuration.

St-DRC [33], which is an elastic DRAM refresh manager, efficient DNN computing utilization, taken as a priority. St-DRC utilizes DNNs' resistance to inconsequential bit errors via the parity bits employed to correct the critical bits that the constant refresh periods lately result in. It incurs energy expenditure for DRAM refresh in both training and inference phases thus preserving the DNN performance. Simulators with symmetric upgrade potential for both graphic and main memories during training have been demonstrated at roughly 23%/12% DRAM energy savings and acceleration of 0.43~4.12% in training time.

The innovative DRAM refresh method resolves the problem of DRAM refresh by delivering read and refresh operations simultaneously. This method is named DR DRAM [34] an efficient hardware architecture that allows DR refreshing. Unlike the conventional approaches, the DR is focusing on updating selected devices, which in return enhances read efficiency by getting the unattainable data. Additionally, Hybrid Refresh Main Memory (HRMM) is introduced that can be customized based on the refresh scheme. Planned benefits include application scalability, covering SPEC CPU2006, CNN, LLT, and PageRank tasks.

Next work [35] is concerning observing a method for creating neural networks deep measurement device examination of test vector. By means of reinforcement machine learning technique, the method takes in stimuli representing the input devices and outputs the corresponding pin numbers for covering the designated circuit. The approach adopted is focused on looking for the best solution using policy gradient approaches with several tools like Å-nearest neighbor search, transfer learning, and replay buffer. The test vectors created have full coverage on fifteen blocks of circuit configuration of the design file with a compress ratio of only 7% as compared to those created by human.

To improve the energy efficiency and performance of DNN inference systems, authors [36] propose using approximation DRAM, which runs with a lower supply voltage and access latency, resulting in significant energy savings and evaluation time. EDEN (Energy-efficient, Deep neural network Engine) is a star of their approach; it is a novel system developed to precisely balance energy efficiency and inference accuracy, all while adhering to the user's preset DNN accuracy. EDEN can also save up to 40% of energy while maintaining the adequate accuracy levels of CNNs and RNNs that are common DNN models. EDEN demonstrates the practical benefits of approximating DRAM in real-world DNN applications, leading the way to a creation of more energy-efficient and high-performance computing systems.

[37] introduces an approximate memory architecture designed to reduce DRAM refresh power consumption in deep learning systems by exploiting their tolerance to minor errors. By organizing data storage in a transposed manner and varying refresh rates based on data significance, the architecture achieves a significant reduction in power usage (69.68%) with

minimal impact on accuracy. Tested on GoogleNet and VGG-16 models, this method offers a promising approach for energy-efficient computing in error-resilient applications.

RTC comes as an [38] approach that is used to increase the energy efficiency of Convolutional Neural Network (CNN) accelerators by optimizing the DRAM refreshes. DRAM refresh can make up to 40% of total energy consumption alone. RTC has two basic ways to eliminate needless refresh operations: access to remote data whenever and wherever RTT is needed and the use of the partial array auto-refresh (PAAR). RTT can skip certain refresh operations due to predicted behavior of CNN’s access patterns. PAAR contrary removes refreshes for the part of DRAM memory that is left with no use at all. Moreover, authors came up with novel method: instead of the traditional way of treating DRAM refreshes, systems have fixed refresh intervals which will mainly focus on weak rows with frequent read operations for data retention. This kind of technology can reduce refresh energy usage to third or even half and drop DRAM energy consumption to 31.8%.

Retention-Aware Neural Acceleration (RANA) architecture for optimizing energy usage in CNN accelerators with Embedded DRAM (eDRAM) is proposed in [38]. Recognizing that eDRAM's periodic refresh cycles for data integrity considerably contribute to energy consumption, RANA takes use of the fact that refreshes are unnecessary when data lives are lower than eDRAM retention times. It includes retention-aware training, dynamic calculation pattern selection using an energy model, and a refresh-optimized eDRAM controller to reduce refresh operations. RANA reduces eDRAM refresh energy by up to 99.7%, resulting in up to 66.2% system energy savings over typical SRAM-based accelerators without sacrificing precision. This method presents a novel solution for increasing the energy efficiency of hardware accelerators for neural networks by tackling the refresh cost in eDRAM storage.

# Chapter 4 – Methodology

To study the impact of DRAM refresh rate on various parameters of Neural Networks including accuracy, power consumption, runtime and general performance the following methodology is proposed. It will be divided into 3 stages: Simulation stage to study the impact of data loss in DRAM on NN performance in terms of accuracy. The hardware simulation stage is devoted to testing NN power consumption and runtime with a reduced DRAM refresh rate on ideal simulated hardware. Finally, hardware implementation stage is devoted to testing NNs on various refresh rates on a physical setup with a real-world environment.

## 4.1. Simulation

To begin the study of Neural Networks' behavior at reduced DRAM refresh rates, we should prove the tolerance of NN to data losses in DRAM. During the inference, Neural Networks have already pre-trained weights, thus only a forward pass is necessary. This entails that NNs store only model parameters and weights in DRAM during inference [39]. Therefore, mimic the data loss in DRAM, weights of the pre-trained NN model can be disturbed. Representing the weights in binary format and flipping random bits with value “1” to “0” during inference will closely simulate the real behavior of leaking DRAM cells. Goal is to evaluate the effect of such produced bit flips which are similar to DRAM leaks on the accuracy of NNs.

For this task, Convolutional Neural Network models were chosen because they are most suitable for computer vision and image recognition tasks that are very popular on devices with limited computational power. Also, CNNs are very popular, have developed frameworks and large sets of pre-trained models.

In total 8 large models were chosen that were trained on the ImageNet dataset: ResNet50, ResNet101, ResNet152, AlexNet, DenseNet121, InceptionV3, MobileNetV3, and WideResnet101. The reason to pick large models is to maximize the number of weights to closely simulate the minor data loss in DRAM.

ResNet or Residual Network is convolutional neural network (CNN) architecture introduced in 2015 by He et al. [40], outperforms traditional CNNs such as VGG. It uses shortcut connections to solve the vanishing gradient problem. ResNet 50 has a bottleneck design with 1x1 convolutions for faster training and consists of 48 convolutional layers, 1 MaxPool layer, and 1 average pool layer. ResNet 101 is an extension of ResNet 50 with more convolutional layers. ResNet 152 is a further scaled version of ResNet 50 with 152 layers.

DenseNet (Dense Convolutional Network) [41] is a variant of ResNet that addresses the vanishing gradient issue by establishing dense connections between layers, so that every layer is connected to every other layer by feed-forward method ensuring maximum information flow. Unlike traditional networks, every layer in DenseNet receives direct inputs from all preceding layers, improving parameter efficiency and facilitating better feature extraction for computer vision tasks.

Another version of ResNet is WideResNet [42] which surpasses ResNet because it can adjust network width without efficiency drops. Introduced in 2016, this technique solved the problem of vanishing gradients in deep networks. WideResNet shows that instead of deepening networks one can also just broaden them and achieve better results. It is Batch Normalization - ReLU - Conv structure, where ResNet blocks contains convolutional groups, which is depending on network width. Tuning settings such as the style of convolution, the number of convolutions per block, the width of residual blocks, and dropout layers can be implemented to meet different needs. Wide ResNet makes it possible to boost the accuracy for image recognition tasks and improves the performance.

AlexNet [43], released in 2012, is by far the deep learning technique that completely reshaped this part of the field by winning the ImageNet Large Scale Visual Recognition Challenge competition. Its architecture has eight layers, including five convolutional layers and three fully connected layers with ReLU activation and dropout regularization. AlexNet adopted the use of GPUs for speeding up the training process through the introduction of local response normalization and overlapping pooling to enhance generalization. It created effective models for deeper neural networks and embraced the application of convolutional neural networks (CNNs) in various vision-related problems.

InceptionV3 [44] is a deep CNN (Convolutional Neural Network) architecture intended for the image classification and recognition tasks. This update of Inception model was done with the aim of improving computational efficiency and accuracy. InceptionV3 also has a unique architecture with multiple parallel convolutional pathways. The pathways are called "inception modules" depending on the ability to detect the objects at various scales. It involves approaches such as batch normalization and factorized convolutions to minimize the contributions and maximize efficacy. InceptionV3 remains popular nowadays as either feature representation or fine tuning for variety of computer vision applications.

MobileNetV3 [45] is a lightweight convolutional neural network (CNN) optimized for mobile and embedded devices that operate with constrained computational resources. It aims to ensure high level of precision and Productivity comparing the other versions. MobileNetV3 utilizes the depth wise separable convolutions, inverted residuals, and linear bottlenecks simultaneously to slim down parameters and computations and still impart good performance. The new architecture has introduced the functionality of hard sigmoid activation function and the squeeze and excite module for better feature representation and model capacity enhancement. MobileNetV3 demonstrated state-of-the-art performance on categories of image classification tasks and has become the standard choice for AI tasks in resource constrained edge devices.

All models are pretrained on ImageNet-1k at resolution 224x224 and accessed from the PyTorch library and simulation tests will be running on Python. The simulation setup is constructed from 5 steps.

The first step is to run a validation for all models using the original weights that were pre-trained on the ImageNet dataset. Next, all the weights are converted to binary. All weights in Pytorch are stored in float32 format, so a single weight is equal to 32 bits of data. All weights are converted to binary format and randomly bits will be flipped from 1 to 0. The random bits in this setup are chosen with rand() function without random seeding. It is explained by the random distribution of weak cells in the DRAM bank [46]. It is not possible to locate the individual weak DRAM cells, thus random function is chosen in this case.

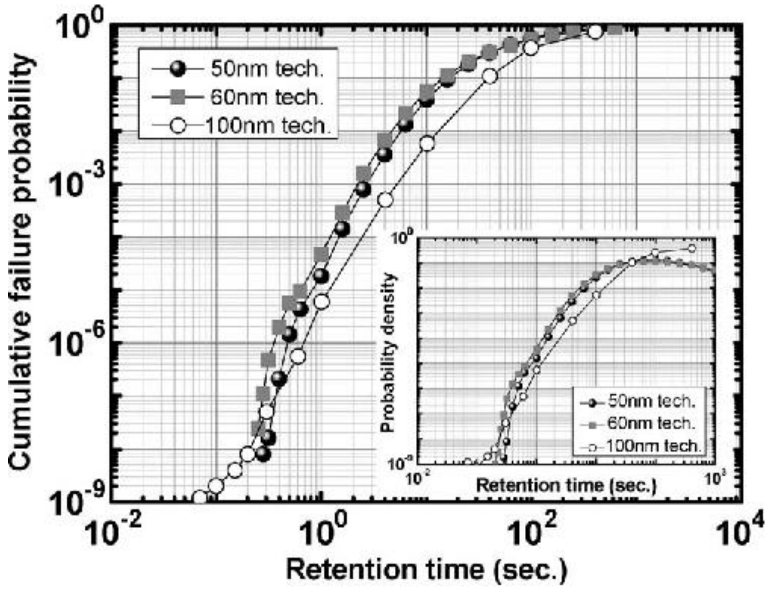


**Figure 5. Retention time for DRAM cells** [47]

After flipping the random bits, all of the weights are converted back to float32 format. Finally, validation is performed for all models again, however this time with modified weights. The number of bits that will be flipped in NN’s weights corresponds to real-world DRAM behavior at reduced DRAM refresh rates [4]. For example, [47] shows that at 4 times the normal refresh rate tREFW =256ms, only 0.00000001% of cells lose charge. At tREFW =256ms, 0.00000218% of cells appear to be weak. Looking at this behavior in Figure 5 it is decided to assign a number of bits to flip to be 10,100,1000 and up to 10000000 to test the limits of NN accuracy. The pseudocode of the setup developed for this experiment can be found in Figure 5.

```
# Pseudocode for Simulating DRAM Refresh Rate Reduction and Bit-Flipping in NN Weights

# Import necessary libraries
Import torch, torchvision models, torchvision datasets, torchvision transforms
Import DataLoader, numpy, struct, random

# Define the number of bits to flip
Set num_bits = 10

# Define the device to use (GPU if available, otherwise CPU)
Set device = "cuda" if CUDA is available else "cpu"

# Define function to validate the model
Function validate_model takes (model, dataloader, device):
    Set model to evaluation mode
    Initialize correct and total counters
    Loop over batches in dataloader:
        Transfer images and labels to device
        Compute model outputs and get predictions
        Update total and correct counters
    Calculate accuracy
    Return accuracy

# Define function to convert a float to its binary representation
Function binary takes (num):
    Convert num to binary and return

# Define function to convert binary representation back to float
Function binary_to_float takes (binary_str):
    Convert binary_str to float and return

# Define function to flip a random bit in a binary string
Function flip_bit takes (binary_str):
    Choose a random '1' bit and flip it
    Return modified binary string

# Define function to flip random bits in model weights
Function flip_random_bits takes (model, num_bits):
    Concatenate all model weights into a single tensor
    Select random indices for bit flipping
    Loop over selected indices and flip bits
    Update model weights with flipped bits
    Return changes made

# Define a list of models to process
Set models to list of tuples containing model function, pretrained weights, and input size

# Load validation data
Set validation_data_path to the path of the validation data
Loop over each model in models:
    Load and transform validation data
    Create a DataLoader for validation data
    Load model with pretrained weights or default weights
    Transfer model to device
    Validate the model with original weights and print accuracy
    Apply bit-flipping to model weights
    Validate the model with modified weights and print accuracy
    Print the change in accuracy and details of bit-flipping
```

**Figure 6. Pseudocode for simulation**

## 4.2. Hardware simulation

Power consumption and Runtime parameters of NN can be obtained by either testing on a real system or testing on a virtual hardware system that closely simulates the behavior of real systems. The hardware simulation root is more suitable for initial proof of concept due to its flexibility and availability. To run validation of Neural Networks in simulation cycle-accurate CPU and DRAM model is needed.

Gem5 [48] is chosen as a platform for CPU simulation, because it supports multiple architectures and CPU models, also with good accuracy. For the DRAM model, the DRAMsim3[49] simulator is chosen due to its speed and accuracy validated on real models. The setup is presented in Figure 7. Unfortunately, none of the current DRAM simulators can simulate the behavior of memory loss at reduced refresh rate, they act as cells with ideal capacitors with no leaking current. Therefore, this stage of the experiment is used to showcase the power consumption and performance gains from reducing DRAM refresh rate. The parameter that will be altered to change the refresh rate tREFW is tREFI. The basic tREFI is 9360ns or 9.36 us. tREFI will be increased exponentially to capture various scenarios of DRAM refresh rate.

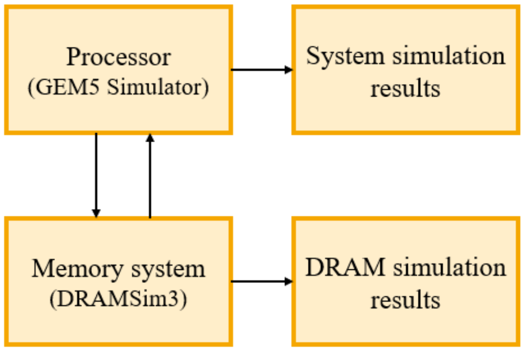


**Figure 7. Hardware Simulation setup**

The setup and configurations are set as in Table 2.

**Table 2. Gem5 and DRAMsim3 parameters**

| Model | Parameters | |
|---|---|---|
| Gem5 | Architecture - X86<br>CPU - AtomicSimpleCPU, 2GHz, 8 core | |
| DRAMsim3 | DRAM: DDR4-8GB-x8-2400 | |
| | tCK = 0.83 | tRRD_S = 4 |
| | tRCD = 17 | tRRD_L = 6 |
| | tRP = 17 | tWTR_S = 3 |
| | tRAS = 39 | tWTR_L = 9 |
| | tRFC = 420 | tFAW = 26 |
| | tREFI = 9360* | tWR = 18 |

In terms of NN, the simulated processor in gem5 is slow compared to the real processor, thus models from section 4.1 cannot be used. Additionally, AtomicSimple CPU supports only syscall binary executables that are generated by C/C++. Following this, the simple single-layer neural network was chosen [50]. This network is pre-trained on MNIST and will be validated on 1000 MNIST images because the ImageNet dataset is too complex for the simulated processor.

## 4.3. Hardware implementation

For the hardware implementation, the Xilinx FPGA Artix-7 [51] platform was chosen due to availability. The reason behind choosing a Field-Programmable Gate Array (FPGA) board is its flexibility, hardware can be adapted to specific computational tasks, making it ideal for prototyping and testing neural networks. Artix-7 board is equipped with an external 1GB DDR3 rank suitable for our research. The FPGA board does not have SoC in it, thus a MicroBlaze soft processor core will be used for executing NNs. Microblaze is a soft processor developed for Xilinx meaning that it will be implemented from FPGA fabric.

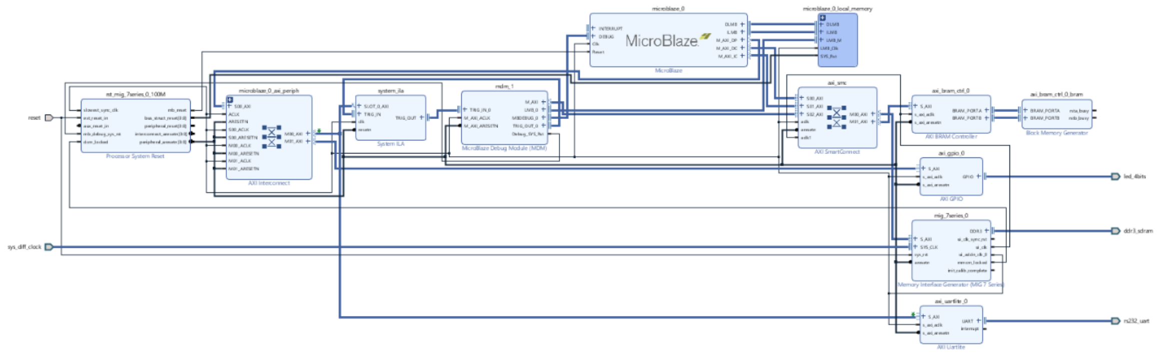


**Figure 8. System block design in Vivado**

DDR3 memory is controlled by the MIG memory controller [52]. MIG uses AXI bus to connect to external memory which in this case is DDR3 rank. Memory controller is responsible for managing the refresh operations by sending REF commands. Modifying the MIG controller to reduce the refresh rate will allow to test NNs in realistic conditions. Development software for Xilinx boards is Vivado HLS, the testing setup is built using the mentioned components and can be observed in Figure 7. The NN will be implemented via Xilinx SDK in C++ language.

# Chapter 5 – Results and Discussion

## 5.1. Results

### 5.1.1 Software

The results from software stage describe the outcome of validation tests that were performed on 8 different Neural Networks using original and modified weights. Each NN model was validated with original weights and with 7 different weights that were modified by flipping 10, 100,1000,10000,100000,1000000,5000000 bits within the weights. Since the fixed amount of bits from overall number of bits present in weight file were flipped, it did not account for different number of weights in each NN. Thus the percentage of flipped bits in each NN model is calculated and then matched to the real DRAM refresh rate according to Figure 5. In Table 3 number of weights for each NN model is presented, along with the number of bits in each weight file. The number of bits are calculated by *NumBits=NumWeights*32* since weights are stored in float32, thus 1 weigth is equal to 32 bits.

**Table 3. Percentage of bits flipped in each NN model.**

| **NN** | **Num weights** | **Num bits** | **% bits flipped** | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 10 | 100 | 1000 | 100000 | 1000000 | 5000000 |
| AlexNEt | 61100840 | 1955226880 | 5.11*E-07 | 5.11E-06 | 5.11E-05 | 5.11E-03 | 5.11E-02 | 2.56E-01 |
| inceptionv3 | 27161264 | 869160448 | 1.15E-06 | 1.15E-05 | 1.15E-04 | 1.15E-02 | 1.15E-01 | 5.75E-01 |
| densenet121 | 7978856 | 255323392 | 3.92E-06 | 3.92E-05 | 3.92E-04 | 3.92E-02 | 3.92E-01 | 1.96E+00 |
| resnet50 | 25557032 | 817825024 | 1.22E-06 | 1.22E-05 | 1.22E-04 | 1.22E-02 | 1.22E-01 | 6.11E-01 |
| resnet101 | 44549160 | 1425573120 | 7.01E-07 | 7.01E-06 | 7.01E-05 | 7.01E-03 | 7.01E-02 | 3.51E-01 |
| resnet152 | 60192808 | 1926169856 | 5.19E-07 | 5.19E-06 | 5.19E-05 | 5.19E-03 | 5.19E-02 | 2.60E-01 |
| wide_resnet101 | 126886696 | 4060374272 | 2.46E-07 | 2.46E-06 | 2.46E-05 | 2.46E-03 | 2.46E-02 | 1.23E-01 |
| mobilenet_v3 | 5483032 | 175457024 | 5.70E-06 | 5.70E-05 | 5.70E-04 | 5.70E-02 | 5.70E-01 | 2.85E+00 |

**Table 4. % of flipped bits and corresponding tREFW.**

| ResNet152 | | Wide ResNet101 | | MobileNetV3 | |
|---|---|---|---|---|---|
| % Bits flipped | tREFW (s) | % Bits flipped | tREFW (s) | % Bits flipped | tREFW (s) |
| 5.19E-07 | 0.63 | 2.46E-07 | 0.41 | 5.70E-06 | 1.10 |
| 5.19E-06 | 0.90 | 2.46E-06 | 0.95 | 5.70E-05 | 1.70 |
| 5.19E-05 | 1.50 | 2.46E-05 | 1.80 | 5.70E-04 | 2.52 |
| 5.19E-04 | 2.32 | 2.46E-04 | 2.00 | 5.70E-03 | 4.77 |
| 5.19E-03 | 4.57 | 2.46E-03 | 3.40 | 5.70E-02 | 11.40 |
| 5.19E-02 | 11.20 | 2.46E-02 | 8.90 | 5.70E-01 | 95.00 |
| 2.60E-01 | 25.20 | 1.23E-01 | 16.20 | 2.85E+00 | 2192 |

**Table 5. % of flipped bits and corresponding tREFW.**

| DenseNet121 | | ResNet50 | | ResNet101 | |
|---|---|---|---|---|---|
| % Bits flipped | tREFW (s) | % Bits flipped | tREFW (s) | % Bits flipped | tREFW (s) |
| 3.92E-06 | 0.64 | 1.22E-06 | 0.48 | 7.01E-07 | 0.46 |
| 3.92E-05 | 1.20 | 1.22E-05 | 0.89 | 7.01E-06 | 0.72 |
| 3.92E-04 | 2.10 | 1.22E-04 | 1.49 | 7.01E-05 | 1.40 |
| 3.92E-03 | 4.10 | 1.22E-03 | 3.10 | 7.01E-04 | 2.47 |
| 3.92E-02 | 9.40 | 1.22E-02 | 6.00 | 7.01E-03 | 5.10 |
| 3.92E-01 | 0.55 | 1.22E-01 | 18.23 | 7.01E-02 | 13.00 |
| 1.96E+00 | 690.00 | 6.11E-01 | 104.00 | 3.51E-01 | 44.50 |

**Table 6. % of flipped bits and corresponding tREFW.**

| AlexNet | | InceptionV3 | |
|---|---|---|---|
| % Bits flipped | tREFW (s) | % Bits flipped | tREFW (s) |
| 5.11E-07 | 0.43 | 1.15E-06 | 0.49 |
| 5.11E-06 | 0.70 | 1.15E-05 | 0.82 |
| 5.11E-05 | 1.30 | 1.15E-04 | 1.47 |
| 5.11E-04 | 2.12 | 1.15E-03 | 3.00 |
| 5.11E-03 | 4.37 | 1.15E-02 | 5.90 |
| 5.11E-02 | 11.00 | 1.15E-01 | 17.65 |
| 2.56E-01 | 25.00 | 5.75E-01 | 89.15 |

Cumulative failure probability in Figure 5 corresponds with % of bits flipped in Table 4,5,6. With this, corresponding tREFW (retention time in Figure 5) can be filled in Tables 4,5,6 for each NN model.

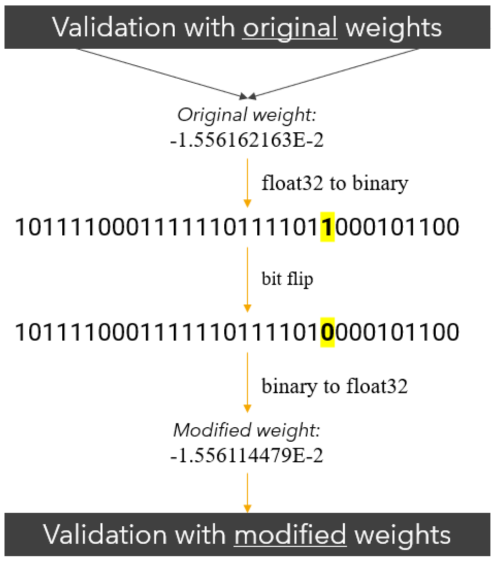
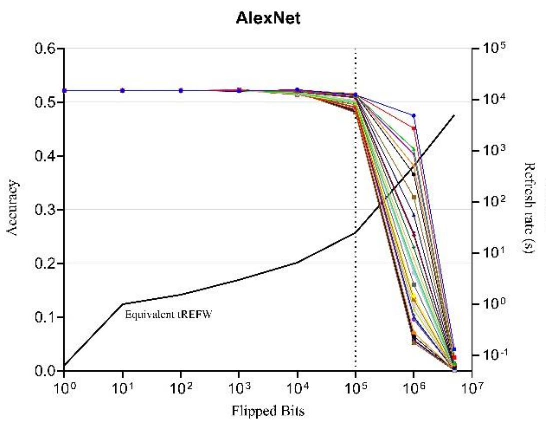
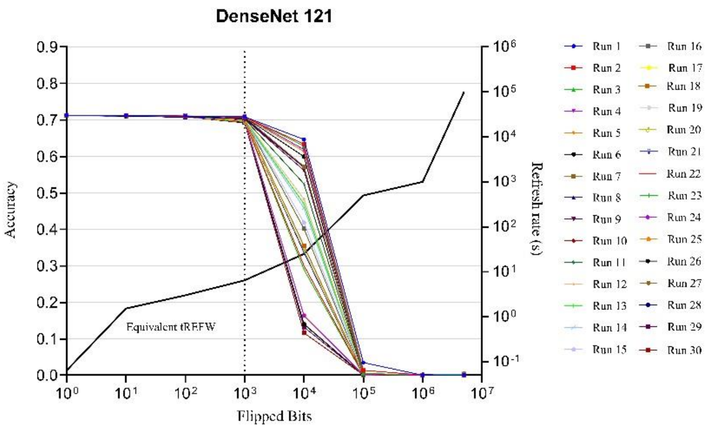


**Figure 9. Simulation flow example for single weight.**

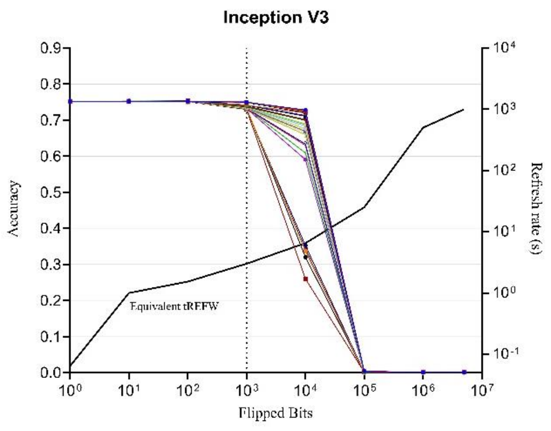
Since the simulation setup uses random function to flip bits, the validation process has been performed 30 times for each model and each set of weights to obtain more credible results. The example of this flow is described in Figure 9. The accuracies obtained from these tests are plotted together with corresponding realistic refresh rates for each NN model.



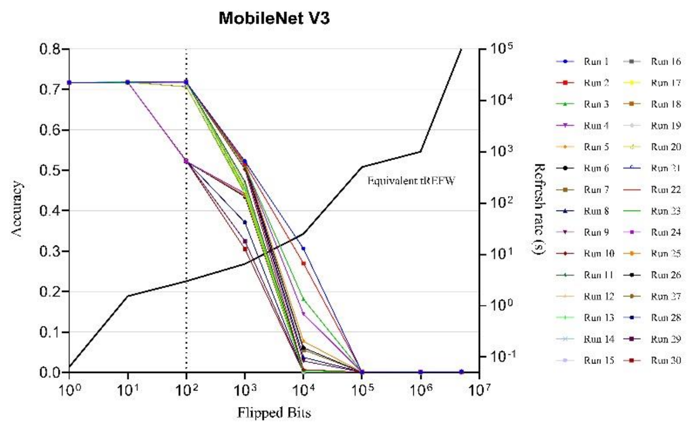
(a) AlexNet



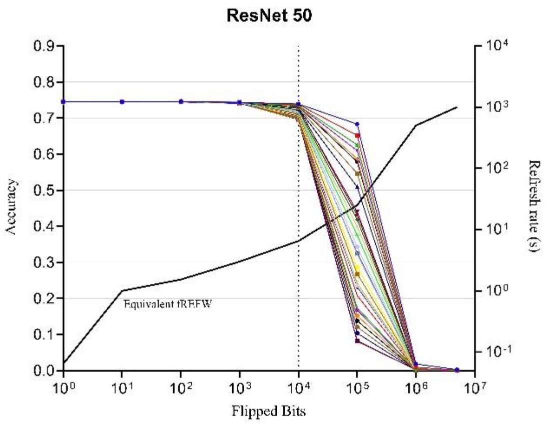
(b) DenseNet121



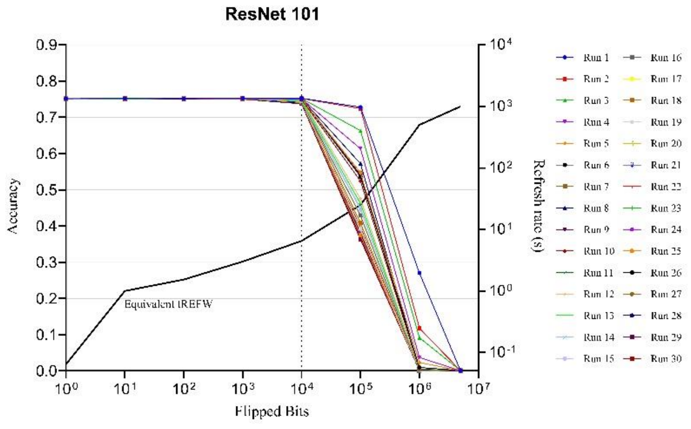
(c) InceptionV3



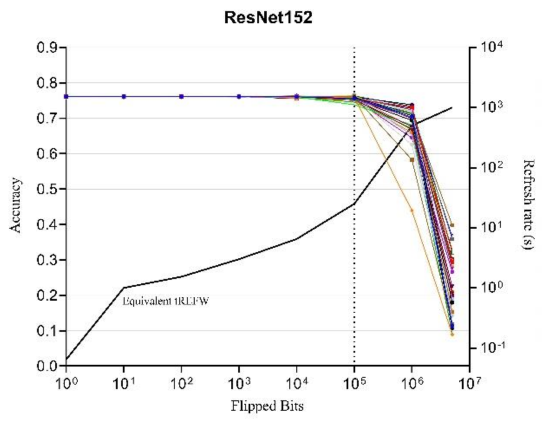
(d) MobileNetV3



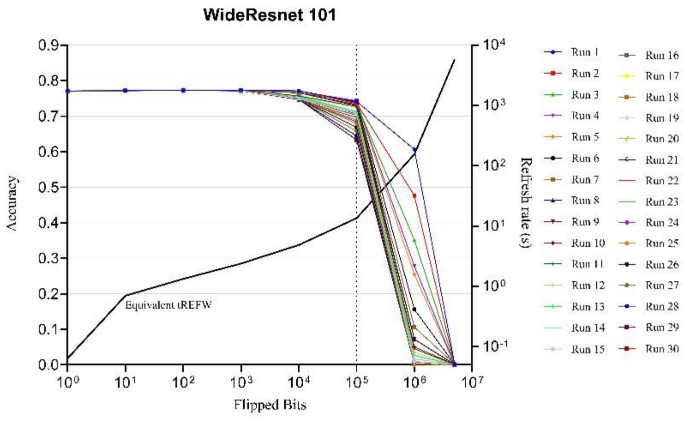
(e) ResNet50



(f) ResNet101



(g) ResNet152



(h) WideResNet101

**Figure 10. Validation accuracies for 8 NN models with tREFW**

To compare the effect of data loss across all neural networks cumulative validation accuracies are plotted in Figure 9. Each NN model has a darker line representing the average value, and all the different values are depicted within the respected colored area.

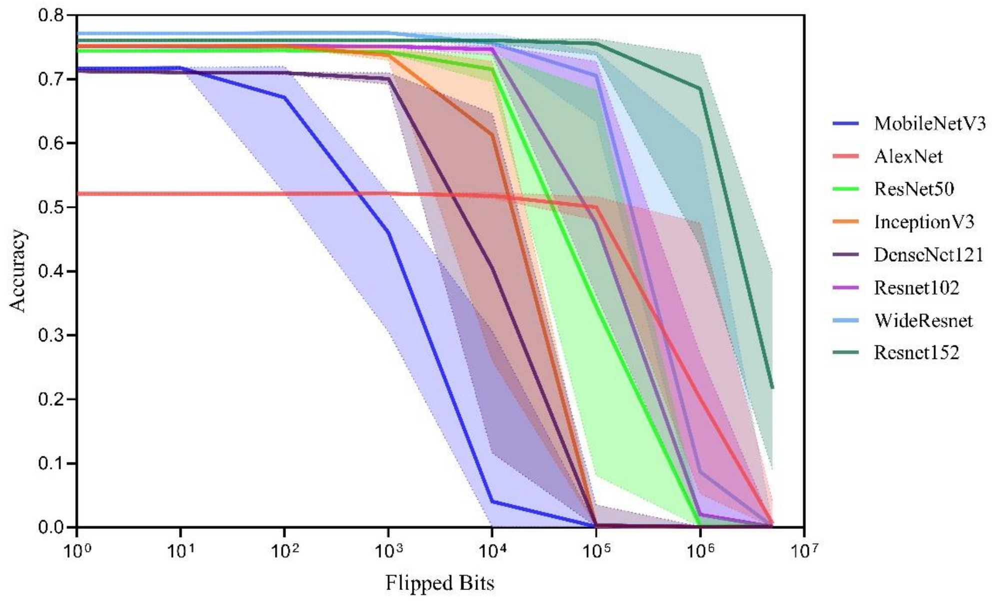


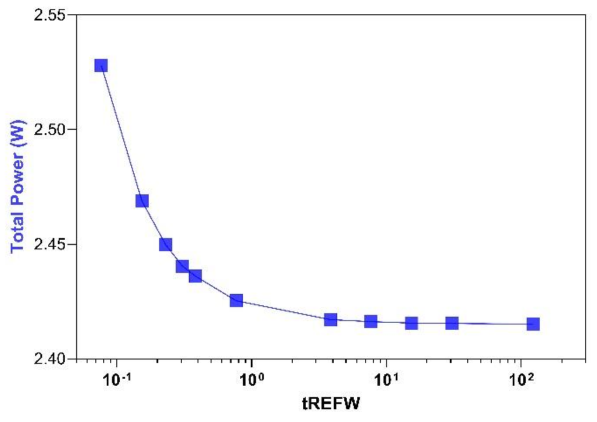
**Figure 11. Cumulative validation accuracies for 8 NN models**

## 5.1.2. Hardware Simulation

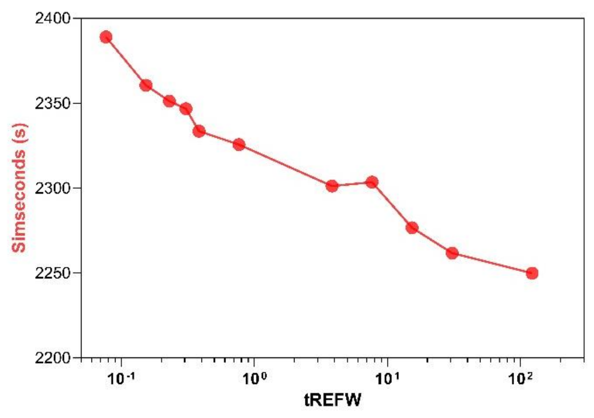
A single layer Neural Network is validated with 1000 MNIST images using a syscall emulation mode in gem5 for various tREFI refresh rate values. To use the results obtained from this experiment tREFW timing needs to be calculated using tREFI. A DRAM rank used for this setup in DRAMsim3 is DDR4-8GB-x8-2400. It has 8 banks and 65536 rows in total across all banks. Also the standart tREFI valuse is 9.36 us. tREFW can be calculated with formula (2.1). Fisrt the number of AREF commands is equal to 65536/8=8192.

$$tREFW = 9.36\ \mu s * 8192 = 76677.12\ \mu s = 0.076\ s \qquad (5.2)$$

Now DRAMsim3 power consumption is plotted in Figure 10a and the time taken to run the validation at each refresh rate is in Figure 10b.



(a) DRAM power consumption



(b) Gem5 simulation time

**Figure 12. Results for hardware simulation**

### 5.1.3. Hardware Implementation

System design proposed in a Methodology section was successfully implemented and FPGA board was functioning as a processor with external DRAM connected to it. Subsequently, simple NN described in Section 4.2 was used for initial testing. The verilog files of MIG memory controller were modified to change the tREFI parameter from standard 7.8us. However, after several validation runs with various refresh rates no difference was observed in system performance or power consumption. It appears to be that the Xilinx proprietary memory controller MIG overrides all the changes that violates the JEDEC standards. According to JEDEC standards [11] DDR3 can have a tREFI timing only in the range of 3.9 us-7.8 us. This standard was set due to the vulnerability of DRAM to cyberattacks. The most notable attack that exploits the DRAM properties is called Rowhammer attack. Rowhammer is an attack that repeatedly accesses the row, sending read/write commands faster than refresh rate so that adjacent rows can have bit flips caused by disturbance errors [53] This attack can gain kernel privileges on a real systems, thus making the manufacturers avoid the bit flips in DRAM. In the case of Xilinx FPGA the only way to reduce the DRAM refresh rate is to create an independent memory controller that supports AXI bus and can be modified despite the JEDEC protocols. Such task is implemented by a group of developers [54] and is out of the scope of this research.

## 5.2. Discussion

Reviewing the simulation results reveals that neural networks are capable of withstanding disturbances brought on by weight data loss. Figure 8 displays many NN models with accuracy levels that remain at the initial level for a large number of flipped bits in weights. It seemed that ResNet 152 was the least susceptible to bit flips; the accuracy began to decline significantly only when 100,000 bits were flipped. MobileNetV3 had the lowest performance in terms of accuracy drop per bit flipped, with accuracy beginning to degrade after 100 flipped bits. This can be explained by the smallest number of overall weights amongst all models chosen in this experiment. Thus, we should look at the accuracy performance with respect to the according refresh rate. Each graph in Figure 8 has a vertical line that signifies the edge adequate accuracy of NN models with respect to flipped bits. This vertical line also intersects the refresh rate tREFW, meaning that this number of bits will be flipped at this tREFW parameter and NN will have an accuracy lying on the vertical line. After studying all NN models it appears that the edge of adequate accuracy happens in the range of tREFW from $10^0$ to $10^1$ seconds. This suggests that the DRAM refresh rate for Neural Network Inference can be reduced by raising the refresh time from tREFW = 0.064s to tREFW = 1-10 seconds, which is 15-150 times faster than the present DRAM refresh rate in most devices.

Next, the results from hardware simulation show the decrease of power consumption as the refresh rate is increased. Also, the simulation time steadily decreases with rising tREFW. There is also an interesting observation in Figure 10a, where power consumption of DRAM stabilizes at about tREFW=5s and keeps steady as the refresh rate is increased.

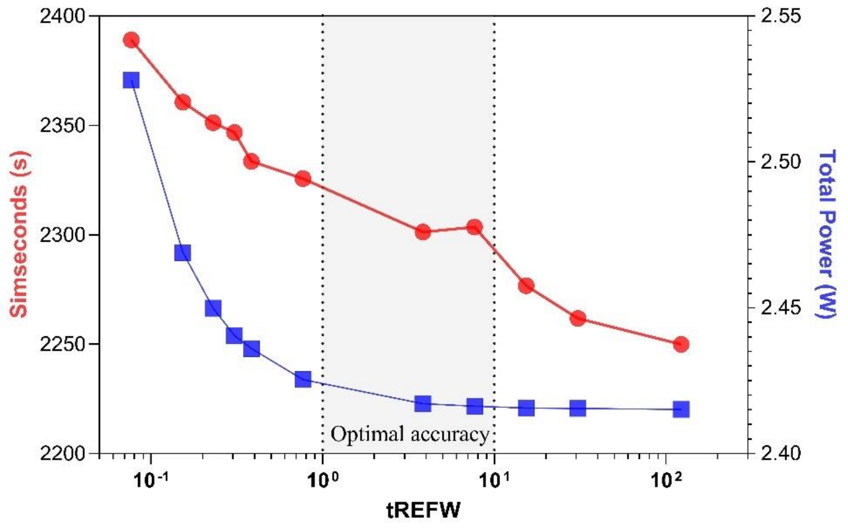


**Figure 13. Power and Simseconds with optimal accuracy**

The results from simulation and hardware simulation were analyzed together. Power and simulation time graphs are combined and overlapped with the outcomes from simulation stage. Figure 11 presents the range of tREFW from $10^0$ to $10^1$ where the optimal NN accuracy was found. At this range there is a noticeable performance improvement in terms of power and runtime. From Figure 11, the minimum tREFW from the range can be chosen to fulfill the accuracy level for all chosen models. Thus, tREFW=1s can be suggested as a safe DRAM refresh rate that can be used for Neural Network Inference. At refresh rate of 1s there is a 2.7% faster inference and 5.6% less power consumed.

Our results show that NNs can withstand minor data errors caused by slower refresh rates with remarkable resilience while preserving performance. We showed thorough simulations and hardware evaluations that it is possible to significantly reduce DRAM refresh rates without sacrificing NN accuracy. Notable gains were made in system performance and energy efficiency due to this decrease in refresh frequency. These findings highlight the viability of using lower refresh rate DRAM in practical neural network applications, opening a promising path for resource-constrained device optimization. The study's conclusions advance ongoing efforts to increase computational efficiency and open the door for more advanced NN models to be used on a variety of platforms. Future research needs to explore the applications of lower DRAM refresh frequencies in real world scenarios, thus extending findings to a larger pool of DRAM technologies and system configurations. Also, it is important to test on variety of devices from embedded systems to large computational machines. Although decreasing the refresh rate poses a safety concern, this method may find application in specialized devices that need just neural network inference, such as image recognition systems.  It is also important to test the compatibility of this method with new DRAM standards like DDR5 and DDR6 in future.

# Bibliography

[1] A. Paleyes, R.-G. Urma, and N. D. Lawrence, “Challenges in Deploying Machine Learning: A Survey of Case Studies,” *ACM Comput. Surv.*, vol. 55, no. 6, Dec. 2022, doi: 10.1145/3533378.

[2] V. Sze, Y. Chen, T.-J. Yang, and J. S. Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” *Proceedings of the IEEE*, vol. 105, pp. 2295–2329, 2017, [Online]. Available: https://api.semanticscholar.org/CorpusID:3273340

[3] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. 2008.

[4] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, “RAIDR: Retention-aware intelligent DRAM refresh,” in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 1–12. doi: 10.1109/ISCA.2012.6237001.

[5] H. Venugopalan, K. Goswami, Z. Din, J. Lowe-Power, S. King, and Z. Shafiq, “Centauri: Practical Rowhammer Fingerprinting.” Apr. 2023.

[6] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler, “Attacking Deterministic Signature Schemes Using Fault Attacks,” Apr. 2018, pp. 338–352. doi: 10.1109/EuroSP.2018.00031.

[7] JEDEC, “DDR4 SDRAM Standard,” 2012

[8] JEDEC Standard JESD79-4A, “Double Data Rate (DDR) Synchronous Dynamic Random-Access Memory (SDRAM) Specification”

[9] D. James, “Recent innovations in DRAM manufacturing,” in *2010 IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 2010, pp. 264–269. doi: 10.1109/ASMC.2010.5551462.

[10] W.-K. Cheng, P.-Y. Shen, and X.-L. Li, “Retention-Aware DRAM Auto-Refresh Scheme for Energy and Performance Efficiency,” *Micromachines (Basel)*, vol. 10, no. 9, 2019, doi: 10.3390/mi10090590.

[11] JEDEC, “DDR3 SDRAM Standard,” 2010

[12] M. Bahi and M. Batouche, “Deep Learning for Ligand-Based Virtual Screening in Drug Discovery,” Apr. 2018, pp. 1–5. doi: 10.1109/PAIS.2018.8598488.

[13] I. Goodfellow *et al.*, “Generative Adversarial Nets,” in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[15] R. K. Venkatesan, S. Herr, and E. Rotenberg, “Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM,” Apr. 2006, pp. 155–165. doi: 10.1109/HPCA.2006.1598122.

[16] C. Isen and L. John, “ESKIMO: Energy savings using Semantic Knowledge of Inconsequential Memory Occupancy for DRAM subsystem.,” in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, Apr. 2009, pp. 337–346. doi: 10.1145/1669112.1669156.

[17] Y.-H. Gong and S. Chung, “Exploiting Refresh Effect of DRAM Read Operations: A Practical Approach to Low-power Refresh,” *IEEE Transactions on Computers*, vol. 65, p. 1, Apr. 2015, doi: 10.1109/TC.2015.2448079.

[18] J. Stuecheli, D. Kaseridis, H. Hunter, and L. John, “Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory,” in *IEEE Internet Computing - INTERNET*, Apr. 2010, pp. 375–384. doi: 10.1109/MICRO.2010.22.

[19] D. Lee *et al.*, “Adaptive-latency DRAM: Optimizing DRAM timing for the common-case,” in *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015*, Apr. 2015. doi: 10.1109/HPCA.2015.7056057.

[20] S. Khan, D. Lee, Y. Kim, A. Alameldeen, C. Wilkerson, and O. Mutlu, “The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study,” Apr. 2014. doi: 10.1145/2591971.2592000.

[21] S. Ghose *et al.*, “What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study,” *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 3, Dec. 2018, doi: 10.1145/3224419.

[22] E. B. Tchernev, R. G. Mulvaney, and D. S. Phatak, “Investigating the Fault Tolerance of Neural Networks,” *Neural Comput*, vol. 17, no. 7, pp. 1646–1664, 2005, doi: 10.1162/0899766053723096.

[23] T. Hirtzlin *et al.*, “Outstanding Bit Error Tolerance of Resistive RAM-Based Binarized Neural Networks,” in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2019, pp. 288–292. doi: 10.1109/AICAS.2019.8771544.

[24] S. Buschjäger *et al.*, “Margin-Maximization in Binarized Neural Networks for Optimizing Bit Error Tolerance,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 673–678. doi: 10.23919/DATE51398.2021.9473918.

[25] K. Zhao *et al.*, “FT-CNN: Algorithm-Based Fault Tolerance for Convolutional Neural Networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1677–1689, 2021, doi: 10.1109/TPDS.2020.3043449.

[26] B. Kim *et al.*, “MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks,” *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 955–967, 2020, doi: 10.1109/TC.2020.2984496.

[27] D.-T. Nguyen, N.-M. Ho, M.-S. Le, W.-F. Wong, and I.-J. Chang, “ZEM: Zero-Cycle Bit-Masking Module for Deep Learning Refresh-Less DRAM,” *IEEE Access*, vol. 9, pp. 93723–93733, 2021, doi: 10.1109/ACCESS.2021.3088893.

[28] A. G. Yağlikçi *et al.*, “HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips,” in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 815–834. doi: 10.1109/MICRO56248.2022.00062.

[29] M. Cheng *et al.*, “TIME: A Training-in-Memory Architecture for RRAM-Based Deep Neural Networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 834–847, 2019, doi: 10.1109/TCAD.2018.2824304.

[30] T.-F. Hsieh, J.-F. Li, J.-S. Lai, C.-Y. Lo, D.-M. Kwai, and Y.-F. Chou, “Refresh Power Reduction of DRAMs in DNN Systems Using Hybrid Voting and ECC Method,” in *2020 IEEE International Test Conference in Asia (ITC-Asia)*, 2020, pp. 41–46. doi: 10.1109/ITC-Asia51099.2020.00019.

[31] Y. Cao *et al.*, “DR DRAM: Accelerating Memory-Read-Intensive Applications,” in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 301–309. doi: 10.1109/ICCD.2018.00053.

[32] X. Lin *et al.*, “ADROIT: An Adaptive Dynamic Refresh Optimization Framework for DRAM Energy Saving In DNN Training,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 751–756. doi: 10.1109/DAC18074.2021.9586265.

[33] D.-T. Nguyen, N.-M. Ho, and I.-J. Chang, “St-DRC: Stretchable DRAM Refresh Controller with No Parity-overhead Error Correction Scheme for Energy-efficient DNNs,” in *Proceedings of the 56th Annual Design Automation Conference 2019*, in DAC ’19. New York, NY, USA: Association for Computing Machinery, 2019. doi: 10.1145/3316781.3317915.

[34] Y. Cao *et al.*, “DR Refresh: Releasing DRAM Potential by Enabling Read Accesses Under Refresh,” *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1584–1596, 2019, doi: 10.1109/TC.2019.2914679.

[35] H. Choi *et al.*, “Application of Deep Reinforcement Learning to Dynamic Verification of DRAM Designs,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 523–528. doi: 10.1109/DAC18074.2021.9586282.

[36] L. Orosa, S. Koppula, K. Kanellopoulos, A. G. Yaglikçi, and O. Mutlu, “Using Approximate DRAM for Enabling Energy-Efficient, High-Performance Deep Neural Network Inference,” 2023, pp. 275–314. doi: 10.1007/978-3-031-19568-6_10.

[37] D. T. Nguyen, H. Kim, H.-J. Lee, and I. J. Chang, “An Approximate Memory Architecture for a Reduction of Refresh Power Consumption in Deep Learning Applications,” Apr. 2018, pp. 1–5. doi: 10.1109/ISCAS.2018.8351021.

[38] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, “RANA: Towards Efficient Neural Acceleration with Refresh-Optimized Embedded DRAM,” Apr. 2018. doi: 10.1109/ISCA.2018.00037.

[39] Frank Denneman, “Training vs Inference - Memory Consumption by Neural Networks,” 2022.

[40] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *CoRR*, vol. abs/1512.03385, 2015, [Online]. Available: http://arxiv.org/abs/1512.03385

[41] G. Huang, Z. Liu, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” *CoRR*, vol. abs/1608.06993, 2016, [Online]. Available: http://arxiv.org/abs/1608.06993

[42] S. Zagoruyko and N. Komodakis, “Wide Residual Networks,” *CoRR*, vol. abs/1605.07146, 2016, [Online]. Available: http://arxiv.org/abs/1605.07146

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[44] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” *CoRR*, vol. abs/1512.00567, 2015, [Online]. Available: http://arxiv.org/abs/1512.00567

[45] A. Howard *et al.*, “Searching for MobileNetV3,” *CoRR*, vol. abs/1905.02244, 2019, [Online]. Available: http://arxiv.org/abs/1905.02244

[46] T. Hamamoto, S. Sugiura, and S. Sawada, “On the retention time distribution of dynamic random access memory (DRAM),” *IEEE Trans Electron Devices*, vol. 45, no. 6, pp. 1300–1309, 1998, doi: 10.1109/16.678551.

[47] K. Kim and J. Lee, “A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs,” *Electron Device Letters, IEEE*, vol. 30, pp. 846–848, Apr. 2009, doi: 10.1109/LED.2009.2023248.

[48] N. Binkert *et al.*, “The gem5 simulator,” *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011, doi: 10.1145/2024716.2024718.

[49] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, “DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator,” *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020, doi: 10.1109/LCA.2020.2973991.

[50] Truong Son Hy, “Neural-Network-MNIST-CPP.” Accessed: Apr. 05, 2024. [Online]. Available: https://github.com/HyTruongSon/Neural-Network-MNIST-CPP?tab=readme-ov-file

[51] Xilinx, “UG952 - AC701 Evaluation Board for the Artix-7 FPGA User Guide (UG952) (v1.4).”

[52] Xilinx, “UG586 - Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions v4.2,” Dec. 2018.

[53] O. Mutlu and J. S. Kim, “RowHammer: A Retrospective,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2020, doi: 10.1109/TCAD.2019.2915318.

[54] H. Hassan *et al.*, “SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies,” in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 241–252. doi: 10.1109/HPCA.2017.62.