Journal of Computer Science

Original Research Paper

DEACT: Hardware Solution to Rowhammer Attacks

[1]Tesfamichael Gebregziabher Gebrehiwot, [1]Fitsum Assamnew Andargie and [2]Mohammed Ismail

[1]*School of Electrical and Computer Engineering, Addis Ababa Institute of Technology, Addis Ababa, Ethiopia*
[2]*Department of Electronics and Communication Engineering, Sasi Institute of Technology and Engineering Andhra Pradesh, India*

Article history
Received: 28-02-2023
Revised: 05-04-2023
Accepted: 15-04-2023

Corresponding Author:
Tesfamichael Gebregziabher Gebrehiwot
School of Electrical and Computer Engineering, Addis Ababa Institute of Technology, Addis Ababa, Ethiopia
Email: tesfamichael.gegziabher@aait.edu.et

Abstract: Dynamic Random Access Memory (DRAM) is a crucial component in modern computing devices. Improvements in process technology have significantly increased the speed and storage capacity of memory devices. However, as memory cells become smaller and closer to one another, annoying circuit disturbance errors such as the Row-hammer problem have become significant. Studies show that attackers can systematically exploit such errors to induce bit flips and take control of local/remote systems. Even though several hardware and software-based mitigation techniques have been proposed, it is still continuing to be a big threat to system security. In this research, we propose DEACT, a counter-based hardware mitigation to the Rowhammer attack. Contrary to existing countermeasures that refresh victim rows or throttle memory access upon excessive row activation, DEACT uses additional row buffers to keep hot rows and prevent further activation. The size of our counter uses 1.67 times less space than the optimal of existing implementations. DEACT not only eliminates the Rowhammer problem, but it also improves the performance of DRAM. We tested DEACT on the TPC and CPU-2006 benchmarks; the average hit rate has increased by 41% when compared to standard DRAM.

Keywords: DRAM, CPU, Rowhammer, Security, Side Channel Attack

Introduction

DRAM technology scaling, increasing the density of DRAM cells, has enabled better performance in modern computers. However, a study by Mutlu (2013) describes that a strong electromagnetic coupling between compact cells aided by a lower noise margin of smaller nodes has intensified the electrical disturbance errors. Another study by Kim *et al.* (2014b) showed how such errors can be amplified; they demonstrated that frequent activation of a row maximizes inter-cell interference which results in data corruption on vulnerable DRAM cells. They also revealed that at least 139 K row activation is needed to cause data corruption on DDR3 modules.

DDR3 modules have been vulnerable to this error since 2010 according to a study by Lanteigne (2016). DRAM manufacturers have been working on the improvement of inter-cell isolation and have initially considered such problems as a simple reliability concern, not a security one. All attempts were unsuccessful and the problem still persists (Kim *et al.*, 2014b; Liu *et al.*, 2013). The severity of box or gain kernel privileges (Seaborn and Dullien, 2015).

The susceptibility to Rowhammer attacks increases with technology scales. The minimum number of consecutive row activation required to induce bit flips on neighboring cells has reduced by more than 10× since the problem became known (Mutlu *et al.*, 2023). The study also reveals that the susceptibility, the number of bit errors, have increased by 500×.

There exist various local and remote-based attacks that target ×86 (Gruss, 2018; de Ridder *et al.*, 2021; Bosman *et al.*, 2016; Tatar *et al.*, 2018; Zhang *et al.*, 2020a; Cheng *et al.*, 2019) and ARM machine (Van Der Veen *et al.*, 2016; 2018; Frigo *et al.*, 2018). The immediate response of some manufacturers (Apple, 2015; HP, 2015) was to increase the refresh rate on DDR3. However, the mitigation was not effective Lanteigne (2016). Others Gautam *et al.*, (2018; 2019; 2020); Yang *et al.* (2016); Ryu *et al.* (2017) proposed an optimization of cell fabrication to prevent electromagnetic interference between cells. The Target Row Refresh (TRR) (Micron, 2015) method is adopted by manufacturers of DDR4.

However, refresh-based mitigation incurs additional performance and energy penalties (Liu *et al.*, 2012). Even so, a recent publication by Frigo *et al.* (2020), has shown that DDR4 modules are still vulnerable to the Rowhammer attack. Similarly, the smash research (de Ridder *et al.*, 2021) went one step further and demonstrated exploitation from JavaScript, without invoking cache management primitives or system calls. Rowhammer is still a big threat to system security as new attack vectors continued to break previous mitigation.

Hence, it is required to provide effective mitigation where the associated performance overhead is minimal.

In this study, we introduce DEACT, counter-based mitigation to the Rowhammer attack. DEACT effectively stops the Rowhammer attack by limiting the number of row activation to a safe threshold value. Our contribution includes (a) The most space-efficient implementation of a counter table. (b) Effective neutralization of Rowhammer by moving frequently accessed rows to dedicated row buffers.

DEACT not only eliminates Rowhammer, but it also performs better than standard DRAM. We tested DEACT using TPC and CPU-2006 benchmarks; it improves the read hit rate on average by 41.16% for all workloads; decreases the read latency by more than 18%. The remainder of this study is organized as follows. We first provide background information on DRAM followed by a review of Rowhammer attacks and/or countermeasures in current literature. We then discuss DEACT and evaluate it.

DRAM is hierarchically organized into ranks, bank groups, and banks; one or more memory ranks are connected to a memory channel. If a system supports N memory channels, the data transfer rate is increased by a factor of N. A memory rank is a 64-bit wide module that contains a set of DRAM chips that are configured as ×4, ×8, or ×16.

For example, in τηε ×8 configuration, 8 physical chips each with a bit-width of 8 (×8) are connected together Fig. 1. Other configurations include, ×4 (16 chips) or ×16 (4 chips). All ranks work independently. However, full parallelism is limited as all ranks connected to the same channel share the same data lines.

A typical DDR5 rank contains 32 banks. A bank contains an array of memory cells where each cell stores a single bit. The memory cells of a bank are further grouped into several sub-arrays. In a typical DRAM configuration, a bank contains 32 subarrays where each subarray is composed of 32 MATs (multiple cell matrices). Each MAT contains 262,144 memory cells (512 rows and 512 columns).

Figure 2 shows a 1T (one transistor) memory cell; it consists of a capacitor that is connected to the sense amplifier through an access transistor. The line that connects the capacitor and the sense amplifier is called a bit line. A group of memory cells that can be activated by the same word line at once are collectively called a memory row. As the gate of all transistors of the given row is connected to the same word line, activating the word line switches on these transistors thereby allowing a charge flow between the capacitors and the respective sense amplifiers via each bit line.

As can be seen in Fig. 2, each bit line of the memory cell is connected to respective sense amplifiers; these sense amplifiers act as a row buffer. When the open page policy is implemented, recently accessed rows are kept in the row buffer. Consecutive memory requests to the same row, row hit, are served from the row buffer at a lower access latency. However, if the requested address is not located in the same row (row conflict), the access latency will increase as additional tasks are performed; to write back row buffer contents to the previously accessed row and sense the charges stored in each capacitor of the new row.

Memory reads are destructive; charges flow from the capacitor to the sense amplifier during sensing. Moreover, capacitors leak charge; the rate of charge leakage varies from cell to cell due to variations in process technology. In order to prevent the integrity of the stored data, each capacitor's charge is restored back to its original level. The restoration process is called a refresh operation. Even though most capacitors of DRAM cells can retain the charge contents for up to 1 sec, DRAM is refreshed every 64ms dictated by a worst-case scenario.

Rowhammer

The physical size of memory cells has significantly decreased due to advances in process technology. Even though the production of smaller cells has resulted in a high density of memory chips, the noise margin of these memory cells is also getting smaller as smaller capacitors hold fewer charges. As a result, we are witnessing circuit disturbance errors which are caused by frequent activation of nearby rows. The cumulative electromagnetic interference between memory cells could cause unexpected bit flips. Such a kind of problem that occurs during one refresh period is called a Rowhammer (Kim *et al*., 2014b).

Kim *et al.* (2014b) showed that during a 64 ms time period, the minimum number of row activation that is required to induce an error is 139 K for the most vulnerable DDR3 in the experiment. The number of DRAM disturbance errors (Kim *et al*., 2014b) varies as the activation interval is varied. The number of errors reaches its peak at an activation interval of 55 ns (equivalent to approximately 1,140,000 activations) and the disturbance error becomes zero at an activation interval of 500 ns which is approximately equivalent to 125,379 activations. In a similar work, Hassan *et al.* (2021) demonstrated that 10 K activation is required to create a Rowhammer on DDR4 devices while it is required only to make 4.8 K activation to create the same problem on LPDDR4.

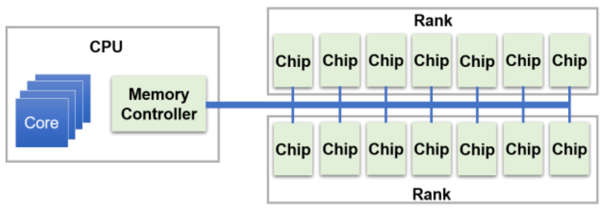

Fig. 1: Memory hierarchy: ×8 Configuration

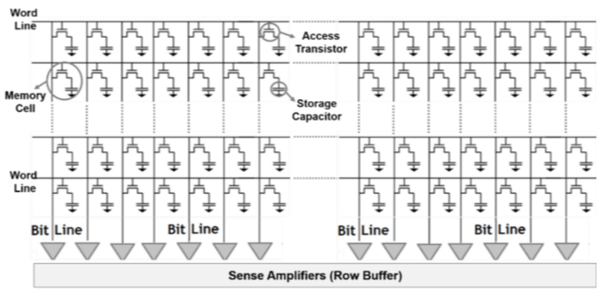

Fig. 2: DRAM bank structure (memory cell organization)

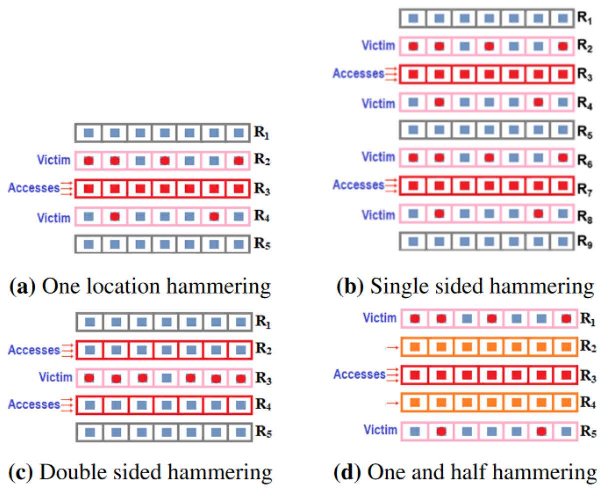
Victim
Accesses
Victim
R1
R2
R3
R4
R5
(a) One location hammering
(b) Single sided hammering
(c) Double sided hammering
(d) One and half hammering

Fig. 3: Hammering techniques

A row that is frequently activated (accessed) is called an aggressor row. Any row whose data integrity is lost due to a hammed row is called a victim row. Figure 3a a single row (R_3) is frequently accessed during a refreshed interval; circuit disturbance errors occur on adjacent rows (R_2/R_4) and the bit value of victim cells is flipped. This hammering technique is called one-location hammering and is only applicable if the memory controller implements a closed-page policy.

If the memory controller implements the open page policy, single-sided hammering is implemented; two rows, located in the same bank, are accessed in an alternating manner. This forces to close previously accessed rows and forces activation by preventing a row buffer. Figure 3b shows how R_3 and R_7 are hammered to induce errors on R_2, R_4, R_6, and R_8.

The most effective hammering technique is double-sided hammering which is shown in Fig. 3c a victim row (R_3) feels the effect of hammering of both aggressor rows (R_2 and R_4). Moreover Fig. 3d shows a one and half hammering technique. When R_3 is accessed frequently while R_2 and R_4 are accessed proportionally fewer times, victims R_1 and R_5 are affected.

Rowhammer Exploitations

There exist various local and remote-based Rowhammer attacks. Modern systems such as mobile devices, servers, and browsers are still vulnerable. The first-Rowhammer exploit, kernel privilege escalation, and escaping browser sandbox, were conducted by Seaborn and Dullien (2015). Other browser exploitation on ×86 machines (Gruss *et al.*, 2016; de Ridder *et al.*, 2021; Bosman *et al.*, 2016) and ARM machines by Frigo *et al.* (2018) are reported. Similarly, kernel privilege escalation was conducted on ×86 machines (Zhang *et al.*, 2020a; Cheng *et al.*, 2019) and ARM machines (Van Der Veen *et al.*, 2016; 2018; Frigo *et al.*, 2018). Tatar *et al.* (2018) achieved the privilege of code execution on a remote key-value server application by implementing attacks through network packets. Rowhammer attacks on Hardware Virtual Machines (HVM) (Razavi *et al.*, 2016) and hypervisors (Xiao *et al.*, 2016) show the extent of threats caused by Rowhammers.

Some forms of Rowhammer attacks (Bhattacharya and Mukhopadhyay, 2016; Kwong *et al.*, 2020; Weissman *et al.*, 2019) work on extracting RSA key; others (Rakin *et al.*, 2022) show how weights of a Deep Neural Network (DNN) can be leaked and yet others show how the accuracy of DNN can be diminished using Rowhammer attack (Hong *et al.*, 2019; Yao *et al.*, 2020). SpecHammer (Tobah *et al.*, 2022), a row of hammer-assisted specter attacks, was able to bypass current specter defenses. Rowhammer can also be the source of Denial of Service (DoS) attacks; (Lipp *et al.*, 2020) implemented network-based attacks on remote systems to compromise system security or cause Denial of Service (DoS). Similar works (Jang *et al.*, 2017; Gruss *et al.*, 2018) showed how DOS attacks can be conducted on local systems.

Rowhammer Mitigation Techniques

DRAM refresh restores the charge in a capacitor and reduces the vulnerability of weaker cells. Following the announcement of the Rowhammer problem, some manufacturers (Apple, 2015; HP, 2015) doubled the DRAM refresh rate from every 64 ms to every 32 ms and claimed to have fixed the problem on DDR3. However, an analysis by Lanteigne (2016), shows that the problem is still prevalent on DDR3 modules.

Many mitigation techniques that focus on refreshing victim rows have been proposed. They are implemented in either DRAM, the memory controller, or both. A few of them are probabilistic (You and Yang, 2019; Kim *et al.*, 2014b; Son *et al.*, 2017); they toss a coin to decide if a row needs to be refreshed. Kim *et al.* (2014a) proposed a work that implements both probabilistic and counter-based target row refresh.

Bennett *et al.* (2021) proposed hardware counters to be implemented in a DRAM MAT. Moreover, much counter-based mitigation has been proposed. They implement different algorithms to count row activation and refresh target rows. Graphine Park *et al.* (2020) implement the Misra-Gries algorithm (Misra and Gries, 1982; Seyedzadeh *et al.*, 2018) employ adaptive tree-based counter, (Lee *et al.*, 2019) uses a TWiCe (time window counter) and many others (Kim *et al.*, 2022; Marazzi *et al.*, 2022; Yağlıkçı *et al.*, 2021; Hong *et al.*, 2023).

Some registered patents implement counters to detect excessive row activation (Devaux and Ayrignac, 2021; Bains and Halbert, 2016; Greenfield and Tomer, 2016; Bains *et al.*, 2015; Gans, 2021; Fisch and Plants, 2017; Greenfield *et al.*, 2014). The only drawback of count and refresh-based mitigation is that increasing the number of refreshes incurs performance and energy penalties.

Several software-based mitigations have also been implemented or proposed. Google (Google, 2014; 2017) updated its Chrome browser to prevent attacks on browser sandbox. Linux (Shutemov, 2015) updated its kernel and restricted access to page map files in order to hide the information about virtual to physical page mapping. Other works that require kernel updates include a work that isolates a user's physical memory location from that of a kernel (Brasser *et al.*, 2017) and a work that isolates the physical memory locations of individual processes (Bock *et al.*, 2019).

Wu *et al.* (2019) introduced a profiling-based mitigation technique. They categorize the nature of error creation in memory cells. They prevented a Rowhammer attack on the page table by placing all page tables on a cell whose value is flipped from 0-1 when an error occurs. Other isolation-based mitigations include Direct Memory Access (DMA) enabled buffer isolation (Tatar *et al.*, 2018; Van Der Veen *et al.*, 2018) on ARM machines and Hardware Virtual Machine (HVM) hypervisor isolation by Konoth *et al.* (2018).

Zhang *et al.* (2022) focus on refreshing DRAM rows that contain page tables whenever abnormal row access patterns are detected. Detecting memory access patterns that are likely to cause bit flips were first introduced by Aweke *et al.* (2016). They tracked abnormal CPU cache misses and suspicious memory access patterns. On the other hand, MASCAT (Irazoqui *et al.*, 2018) performs static analysis on binary code to identify instructions that can cause Rowhammer attacks. Zhang *et al.* (2020b) use radio to control Electromagnetic (EM) signals and detect Rowhammer attacks. Other hardware-based Rowhammer detection methods include work by Gomez *et al.* (2016). They use a dummy cell, a cell with a larger leakage current, to enable early detection of bit flips. Another detection method is implemented by Vig *et al.* (2018) using a sliding window protocol and a dynamic skewed hash tree. Hong *et al.* (2023) employ an approximate counting algorithm to detect hot rows.

Whenever potentially dangerous memory access is detected, some relocate it (Taouil *et al.*, 2021) or throttle it (Yağlıkçi *et al.*, 2021; Greenfield *et al.*, 2015); while others correct bit errors using Error Correction Codes (ECC) (Nair *et al.*, 2016; Ryan and Lin, 2009). It is important to note that ECC cannot correct errors if the number of bit flips exceeds the maximum number of bits that ECCs can correct.

A rather different approach that involves Rowhammer mitigation via fabrication process optimization includes, a work by Yang *et al.* (2016) using additional Phosphorus (P) implantation between two adjacent buried word lines and Ryu *et al.* (2017) using silicon migration technique of hydrogen (H_2) annealing. Gautam *et al.* (2018; 2019; 2020) proposed three works that target the reduction of leakage currents between cells. The first work (Gautam *et al.*, 2018) introduces metal nano-particles at the gate metal-oxide interface; the second work (Gautam *et al.*, 2019) introduces a Metal Nano Wire (MNW) at the gate metal/gate oxide interface. Both techniques induced Energy Valleys (EVs) between nodes to prevent the diffusion of electrons from being hammered to the victim cell. The third work (Gautam *et al.*, 2020) provides isolation between the storage capacitor and the word line that passes over it. The electron current density near the Shallow Trench Isolation (STI) was reduced by 92% when accessing PWL.

Vendors implement the TRR (Micron, 2015) on DDR4 modules. Even though the implementation details are not yet made public, it is based on refreshing target (victim) rows. However, a recent publication by Frigo *et al.* (2020), has shown that DDR4 modules are still vulnerable to the Rowhammer attack. The SMASH research (de Ridder *et al.*, 2021) also demonstrated Rowhammer exploitation on DDR4 from JavaScript, without invoking cache management primitives or system calls.

DEACT

In this section, we discuss our solution to the Rowhammer problem and explain why it is better as compared to other countermeasures.

Why a New Approach?

Manufacturers initially doubled the refresh rate to prevent Rowhammers. However, any solution that tends to double the refresh rate or refresh victim rows incurs significant performance and energy overhead.

Liu *et al.* (2012) showed that the charge retention capacity of DRAM varies from cell to cell and the majority of the cells can retain the charge for a significantly longer duration. Unfortunately, the default refresh rate (64 ms) is set by a few weaker cells which can hold the charge for 64 ms only.

We believe studies should focus on minimizing DRAM refresh rates. Any work that adds extra refresh on top of a refresh rate (dictated by the worst case) could be regarded as inefficient. Hence, we introduce a DEACT, a novel approach that does not perform any extra refreshes. DEACT controls row activation and buffers hot rows for better performance and Rowhammer prevention.

High-Level Overview

The key idea of DEACT is to prevent Rowhammer by detecting unsafe memory access patterns that are likely to cause bit flips; In order to detect such access patterns, DEACT maintains a list of memory rows along with their activation count which can be achieved by implementing a counter. DEACT also implements additional row buffers to keep hot rows from further activation improving the row buffer hit rate.

DEACT can be configured to set the maximum number of activation allowed; any rows that are activated beyond the threshold value are moved to a row buffer dedicated for this purpose. Moreover, the size of the counter and the expiry time of its contents is configurable. The number of row buffers that are needed for this purpose depends on the number of times a Rowhammer can be performed within one refresh period.

Figure 4, DEACT intercepts any ACT (activate row) command and checks if the row address is already in the counter. If present, the value of its activation count is incremented; otherwise, the row address is managed as per the counting algorithm. Word line activation per refresh interval is counted and if the activation count exceeds a thresh hold value, the target row is moved to one of the extra row buffers dedicated for this purpose. The counter is invalidated periodically and its contents are cleared when the expiry time is reached.

In order to track row activation, we need a table that keeps track of all row activation; keeping track of a big list of rows incurs a huge performance, energy, and storage overhead. One way to work around this problem is to maintain a fewer list of row addresses using probabilistic data structures. The count-min sketches (Cormode and Muthukrishnan, 2005) and their derivatives (Ting, 2018; Zhang *et al.*, 2014a) have been widely adopted in detecting heavy hitters of data streams. The accuracy of the count-min sketch can be improved by setting the maximum error rate to a lower value.

Another alternative that deterministically detects heavy heaters is space saving (Metwally *et al.*, 2005) or the Misra-Greis algorithm (Misra and Gries, 1982). The Misra-Greis algorithm, shown in algorithm 1, finds all rows with at least n/k activation using k counters. We maintain a counter table that can hold k-row addresses along with their activation counts. This table is invalidated at a specific time interval (window); the default being every 64 ms (refresh interval). During this interval, the Misra-Greis algorithm (Misra and Gries, 1982) guarantees that all rows which are activated at least n/k times are kept in the table:

Algorithm 1: Misra-Gries algorithm

```
  Procedure Misra-Gries($s$, size) ▷ s a stream sequence
  of positive integers and s is the size of the counter
      $D \leftarrow$ Dictionary [Key, Count]
      while $s \neq$ empty do
          $k \leftarrow s$
          if $D[k] \neq$ empty then
              $D[k] \leftarrow D[k]+1$
          else if $|D| \leq$ size then
              $D[k] \leftarrow 1$
          else
              for $i \leftarrow 0$, size - 1 do
                  $D[k] \leftarrow D[k] - 1$
                  if $D[k] = 0$ then
                      $D[k] \leftarrow$ empty
                  end if
              end for
          end if
      end while
end procedure
```

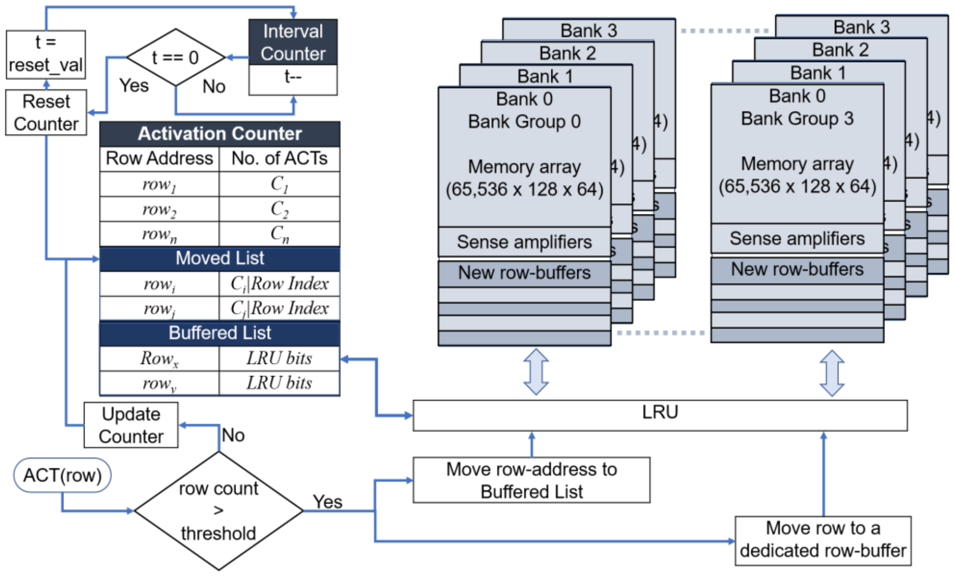

Fig. 4: DEACT: Hardware-based solution to the Rowhammer problem

Counter Table

In order to have an estimate of the maximum number of activations that can be performed within refresh intervals, we look at a specific DRAM model shown in Table 1. DRAM cell is refreshed once every t_{REFW} and a refresh command is invoked every t_{REFI}. The refresh operation takes a time of t_{RFC} and the memory is not available to serve user requests during this time. The attacker can use the remaining t_{REFI} -t_{RFC} time period to induce bit flips. As only one activation can be made within a time period of t_{RC}, the maximum number of row activation that can be made on any bank during a fraction of the refresh window (t_{REFW}/x) is shown in Eq. 1:

$$N_B = \frac{t_{REFW}/x}{t_{REFI}}\left(t_{REFI} - t_{RFC}\right)\frac{1}{t_{RC}} \tag{1}$$

where:

- x is the Reset interval of the counter per refresh window
- Other parameters are described in Table 1

Unlike existing counter-based mitigation (Park *et al.*, 2020; Seyedzadeh *et al.*, 2016; 2018; Lee *et al.*, 2019), DEACT follows a different approach to minimize the space and area overhead. We focus on the DRAM rules that dictate activation per rank. Since a maximum of four activations are allowed during a time period of t_{wo} (four activation windows), we base our computation on t_{FAW} and t_{REFW}. Eq. 2 shows, the total number of activation per rank during a fraction of the refresh window (t_{REFW}/x).

$$N_R = \frac{4}{t_{FAW}}\frac{t_{REFW}}{x} \tag{2}$$

where:

N_R = The number of maximum activation per rank
x = Reset interval of the counter per refresh window other parameters are described in Table 1

As DEACT periodically clears its counter and if we divide the reset window into x time frames of the refresh window (t_{REFW}), the counter is cleared at the end of every t_{REFW}/x time window. Hence, the actual activation count could be less than the real cumulative activation of the current and previous x time frames. Moreover, a victim row could be hammered twice by two aggressor adjacent rows; one from above and another from below (double-sided hammering). In order to compensate for both the double-sided hammering and untimely clearance of rows in the counter table, the Rowhammer threshold needs to be adjusted by a factor of $2(x + 1)$ as shown in Eq. 3:

$$A_{TH} = \frac{R_{TH}}{2(x+1)} \tag{3}$$

where:

A_{TH} = Adjusted activation threshold
R_{TH} = Rowhammer threshold
x = Reset interval of the counter per refresh window

In order to count up to A_{TH} activation, the size of the counter should be at least $\frac{N}{A_{TH}}$, where A_{TH} is the adjusted activation threshold. Equation 4 shows a table size computed using the t_{RC} timing parameter while Eq. 5 shows a size computed using the t_{FAW} timing parameter. For any value activation threshold A_{TH}, the ratio of t_{RC} based vs t_{FAW} based table size is shown in Eq. 6:

$$Table\,Size_{RC} = Num_{Banks}\frac{NB}{A_{TH}} \tag{4}$$

where:

N_B = The number of activation per bank
Num_{Banks} = The number of banks per rank
A_{TH} = The adjusted activation threshold

$$Table\,Size_{FAW} = \frac{N_R}{A_{TH}} = \frac{1}{A_{TH}}\frac{4}{t_{FAW}}\frac{t_{REFW}}{x} \tag{5}$$

where:

N_R = The number of maximum activation per rank
A_{TH} = The adjusted activation threshold
x = Reset interval of the counter per refresh window other parameters are described in Table 1

$$\frac{Table\,Size_{RC}}{Table\,Size_{FAW}} = Num_{Banks}\frac{t_{FAW}}{4t_{RC}}\left(1 - \frac{t_{RFC}}{t_{REFI}}\right) \tag{6}$$

where:

- Num_{Banks} is the number of banks per rank
- other parameters are described in Table 1

$$\frac{Table\,Size_{t_{REFW}}}{Table\,Size_{t_{REFW/x}}} = \frac{2x}{x+1} \tag{7}$$

where:

t_{REFW} = Refresh time window
x = Reset interval of the counter per refresh window

Substituting the values of the timing parameters of the DRAM model shown in Table 1 in Eq. 5 yields 1.13, 1.84, and 1.27 for ×4, ×8, and ×16 configurations. On average t_{FAW} based estimation reduces the space overhead by a factor of 1.4. The space can be further reduced by decreasing the reset interval; i.e., the default 64 ms (t_{REFW}) is divided into x time frames. Taking the ratio of $t_{REFW}/1$ and t_{REFW}/x in Eq. 5 yields Eq. 7. Reducing the reset interval from t_{REFW} with $t_{REFW}/2$, reduces the space overhead to ¾ of its original value.

Table 1: Key timing parameters of DDR4-2400P (JEDEC, 2021)

Parameter	Description	Configuration ×4	×8	×16
t_{RC}	Row cycle	45 ns	45 ns	45 ns
t_{RFC}	Refresh cycle	350 ns	350 ns	350 ns
t_{REFI}	Refresh interval	7.8 µs	7.8 µs	7.8 µs
t_{REFW}	Refresh window	64 ms	64 ms	64 ms
t_{FAW}	Four activation windows	3.33 ns	21.67 ns	30 ns

DEACT implements the counter table at the Register Clock Driver (RCD); it also keeps a list of buffered rows and implements LRU as a replacement policy. The task of inserting row addresses and updating activation counts is done independently of any DRAM operation. The memory controller also needs to be informed whenever a row is added or evicted from a row buffer. This way the memory controller knows what command to issue (PRE, ACT, or RD/WR) when targeting a specific row. Therefore, a new DRAM command (BFR) is sent to indicate if a row is active (buffered) (1) or not (0).

Rowhammer Prevention

For the DRAM model shown in Table 1, the maximum number of row activation of a bank during one refresh window (using Eq. 1) is approximately 1358405 (1.3584 M). This value is much higher than the minimum number of activation required to induce bit flips (10 K) Hassan *et al.* (2021). Recall that the number of activation required to induce bit flips is 39 K Kim *et al.* (2014b) on DDR3 devices, 10 K (Hassan *et al.*, 2021) on DDR4, and 4.8 K on LPDDR4.

Rowhammer can be prevented as long as the number of row activation is kept below the Rowhammer threshold. Once excessive activation is detected, hot rows are moved to a row buffer dedicated to this purpose. The quantity of these buffers, which are located in each bank, is determined by the number of hot rows. A row is called hot if it is activated beyond the Rowhammer threshold value. The number of hot rows can be computed using Eq. 8 note that we have divided the Rowhammer threshold value by 2 to compensate for the effects of double-sided hammering. In order to compute the maximum number of possible hot rows:

$$HOT_{ROWS} = \frac{\left(t_{REFI} - t_{RFC}\right)}{R_{TH}/2}\frac{t_{REFW}}{t_{REFI}}\frac{1}{t_{RC}} \tag{8}$$

where:

- HOT_{ROWS} is the maximum number of possible hot rows
- R_{TH} is the Rowhammer threshold
- Other parameters are described in Table 1

For a Rowhammer threshold of 10 K and a DRAM model shown in Table 1, the number of hot rows is approximately 272. The sense amplifiers that make up a row buffer require more than 100 × space as compared to normal memory cells. It would incur a huge area overhead to allocate 272-row buffers per rank. Moreover, the Rowhammer threshold may continue to reduce as a result of technology scaling. Hence, it is required to minimize the number of row buffers as much as possible. Instead, we could use a mix of sense amplifiers and standard memory cells; we call this a safe area.

Implementation of DEACT is shown in Fig. 4; once a memory row in the main area starts to be hot, it is moved to one of the row buffers in the safe area. Subsequent memory requests to the same row are served from the row buffer. However, if a different row becomes hot and all row buffers are occupied, we apply the Least Recently Used replacement (LRU) policy to evict a row and replace it with the new highly activated row. The evicted row remains in the memory cells of the safe area until the refresh window is elapsed. During this period, the counter table is updated to hold the index of the row in the safe area. If any row in the safe area is activated more than the Rowhammer threshold size of the safe area, it is then buffered again. The benefits of our approach are double-fold as the extra row buffers improve the performance of DRAM.

Materials and Methodology

We used a machine equipped with Intel CORE i7, four 2.5Ghz logical processors with 12 GB RAM to run DDRSharp, a cycle-accurate DRAM simulator (Gebrehiwot *et al.*, 2023), to evaluate DEACT using CPU traces; these traces are made available by the SAFARI research group at ETH Zurich and Carnegie Mellon University (Kim *et al.*, 2015). We compared DEACT with DDR4-2133R (JEDEC, 2021) using traces of the CPU2006 (Henning, 2006; TPC, 2023) benchmarks; each workload is simulated for 1 billion cycles. The basic configuration settings of DDRSharp used for this evaluation are shown in Table 2.

Table 2: Configuration parameters

Component	Parameter	Value
	Number of cores	1
	Frequency	3.2GHz
CPU	ROB size	128
	ROB fetch/retire width	3
	MSHR size	32
	Read/write queue size	64
	Scheduling policy	FRFCFS
Memory	Refresh policy	Rank
Controller	Page Policy	Open
	Channels	1
	Ranks	1
DRAM	Bank groups	4
	Banks per bank group	4
	row buffers per bank	8
	activation permitted	2
DEACT	Validation interval	32 ms
	entry size of counter	64

Evaluation

Rowhammer is prevented by always keeping the number of row activation below the activation threshold. In order to achieve this, we must detect all excessive activation and buffer them. As long as the counter guarantees that all n/k activation is detected, we can assert that no circuit disturbance errors can be induced. DEACT implements the Misra-Gries algorithm (Misra and Gries, 1982).

To prove the correctness of the Misra-Gries algorithm, we describe how the algorithm works. A Table (T) counts instances of row activation R_i; if there is free space in T and R_i is not recorded yet, R_i is added with a counter value of 1 to T. If R_i is already stored in T, its counter value is incremented. However, when T is full and R_i is not in T, R_i is discarded and the count value of each item in T is decremented by 1. Any row in T whose count value is 0 is discarded from the list.

Let C_r be the estimated count of row r, S stream of row addresses and F_r be the actual frequency of r.

Claim: For every $(r, C_r) \in S$, $F_r - n/k \leq C_r \leq F_r$.

Proof: To prove that all elements with frequency at least n/k will have a non-zero counter at the end, let X be an occurrence of r which is discarded and Y be an occurrence of r which is decremented. Therefore, the count value of row r is given by:

$$C_r = F_r - X - Y$$

Table 3: Increase in hit rate and throughput

	Hit Rate		Throughput	
Input	Read (%)	Write (%)	Read (%)	Write (%)
403.gcc	87	1750	1.45	1.66
447.dealII	32	240	0.47	0.84
464.h264ref	70	74	7.46	7.40
481.wrf	60	219	0.05	0.05
tpch6	33	66	3.44	3.42
tpch2	20	39	10.16	10.21
tpch17	21	49	10.90	10.73
tpcc64	9	79	4.85	5.11
Average	42	314	4.85	4.93

Table 4: Latency reduction

Input	Read (%)	Read queue (%)	Write (%)	Write queue (%)
03.gcc	32.2	50.1	35.9	42.9
447.dealII	22.8	44.1	43.3	51.4
464.h264ref	22.5	40.9	79.8	83.4
481.wrf	30.7	55.5	55.0	63.9
tpch6	14.3	17.5	18.8	19.0
tpch2	9.8	11.9	15.5	15.7
tpch17	11.9	14.5	17.4	17.5
tpcc64	7.4	10.9	22.1	22.5
Average	18.9	30.7	36.0	39.5

Table 5: Decrease in activation energy reduction

Input	Activation energy (%)
403.gcc	52.6
447.dealII	48.9
464.h264ref	58.0
481.wrf	54.5
tpch6	47.1
tpch2	41.8
tpch17	47.2
tpcc64	18.2
Average	50.0

With k counters, the number of times that a discard and/or decrement can occur is at most an n/k fraction of the total stream length (n). Hence, $X + Y \leq n/k$; then we have:

$$F_r - n/k \leq C_r \leq F_r$$

Performance Evaluation

Even though DEACT was designed to prevent Rowhammer, it also improves the performance of DRAM. Experimental results of 1 billion cycles simulation time show very high hit rates and lower access latencies. As a result of the increased hit rate, DEACT has performed more reads and more writes than the standard DRAM. Memory requests that would otherwise have caused row buffer conflict in standard DRAM are immediately served by DEACT buffers without the need for row activation; hence higher throughput. Table 3 shows the increase in hit rate and throughput of CPU2006 benchmarks (Henning, 2006; TPC, 2023) workloads. The read/write throughput has increased by more than 10% for the TPC (2023) workload.

While the write-hit rate of 403. gcc workload has increased by more than 1700% (from 0.02-0.37%), and the average increase in hit rate, for all workloads, is 41.16% for reads and 314.35% for writes.

The average latency for both read and write requests have decreased significantly. Table 4 shows the average memory access latency and the average latency of each request on the queue. The average write latency for the 464. h 264 ref workload has decreased by 79.8%. The average write queue latency for the same workload has decreased by 83.4%. For all workloads, the average decrement in read or write latency is 18.9 and 36% respectively while the queuing latency, for read queue and write queue, has on average decreased by 30.7 and 39.5% respectively.

The activation energy of DEACT for each workload is significantly lower than standard DRAM. Yet DEACT was able to perform more reads and more writes with fewer activation when compared to standard DRAM during the 1 billion simulation cycle. Table 5 shows an average of 50% reduction of activation energy.

Area Overhead

The counter table is maintained at the RCD and the number of bits required per rank is equal to $\log_2 N_b$anks + $\log_2 N_r$ows. For a DRAM configuration (×8) specified in Table 1, a rank with 16 banks and 64 k rows per bank, the total number of bits required per entry is given by $\log_2 16+\log_2 64k = 20$ bits. We also set the table reset interval at ½ of the refresh interval which is 32 ms and τηε ×8 DRAM configuration. The number of LRU bits depends on the number of row buffers per bank. Eight-row buffers require 3 bits per bank. Therefore, for a DDR4 rank with 16 banks, the total number of LRU bits is then $\log_2 (8\times16) = 7$. We also need 1 additional bit to indicate if a row is buffered or is moved to a safe area.

Counter Overhead

For evaluation purposes, like most existing mitigation (Park *et al*., 2020; Seyedzadeh *et al*., 2016, 2018; Yağlıkçi *et al*., 2021), we set the Rowhammer threshold at 32 K. Using Eq.2, the maximum number of activation per rank than can be conducted during one refresh period is 5,906,784. Setting the value of × to 2 in Eq. 3 yields, $A_{TH} = 5.33$ k. To count up to 5.33 k, we need 13 bits; a total of 34 bits are required per single entry. Using Eq. 3 and 5, we get a table size of 1108 entries. As each entry requires 34 bits, the total size required per rank is therefore (37656 bits + 7 LRU bits) (4.71 kB).

Overall DEACT needs 4.71kB at the RCD for 32 k activation threshold. This is very low compared to other counter-based mitigation. Table 6 shows a detailed comparison between DEACT and existing works. When compared to the block hammer (Yağlıkçi *et al*., 2021), for the same activation threshold, DEACT reduces the storage requirements by a factor of 11.64.

Recent studies show that the minimum activation required to induce bit flips is 10 k (Hassan *et al*., 2021) which is significantly lower than 32 k. Using Eqs. 2, 3, and 5, the size of the counter table increases to 113,417 bits (14.18 kB). Therefore, the total area overhead at the RCD using a 29.2 Mb/mm^2 SRAM fabricated using 7 nm CMOS FinFET technology (Yokoyama *et al*., 2020) is approximately 109,895/29.2 Mb/mm^2 = 0.0039 mm^2 for 10 k activation threshold. However, DEACT is implemented using a 1.8 Mb/mm^2 TCAM (Tsukamoto *et al*., 2015) which increases the area overhead at the RCD by around 0.063 mm^2.

Table 6: Comparison of space overhead per memory rank

	Overhead (kB)	Ratio
DEACT	4.57	-
Graphene. Park *et al*. (2020)	7.62	1.67
Block hammer. Yağlıkçi *et al*. (2021)	53.21	11.64
TWiCe. Lee *et al*. (2019)	37.12	8.12
CBT. Seyedzadeh *et al*. (2016; 2018)	24.50	5.36

Overhead at DRAM Bank

Typically, a bank consists of 32 sub-arrays where each sub-array contains 32 MATs; each mat is composed of 512×512 memory cells and one local row buffer (Zhang *et al*., 2014b). For this particular example, the total number of memory cells per bank is 512×512×32×32 = 268,435,456. Similarly, the total number of sense amplifiers per row buffer is 512×32. The total number of sense amplifiers per bank is therefore equivalent to 512×32×32 = 524,288.

DEACT implements 256 memory rows and 8-row buffers per bank. That is 8-row buffers ×512×32 (131, 072) sense amplifiers per row buffer. As a row contains 512×32 memory cells, a total of 256 × 512 × 32 which is equal to 4,194,304 cells are implemented. A sense amplifier is 100× larger than a memory cell (Chang *et al*., 2016), Eq. 9 computes the estimated area head per DRAM bank which is 5.39%. Data movement within a bank is performed by implementing LISA (Chang *et al*., 2016) at a cost of only 0.8% DRAM area overhead. In total, DEACT consumes 6.2% of the DRAM area and an area of 0.0603 mm^2 at the RCD:

$$Overhead = \frac{\left(100\times DEACT_{SA} + DEACT_{MC}\right)}{DRAM_{MC} + 100\times DRAM_{SA}} \qquad (9)$$

where:

$DEACT_{SA}$	=	The number of sense amplifiers implemented by DEACT
$DEACT_{MC}$	=	The number of memory cells implemented by DEACT
$DRAM_{MC}$	=	The number of memory cells within a DRAM bank
$DRAM_{SA}$	=	The total number of sense amplifiers within a DRAM bank

Energy Overhead

The static energy overhead on 1.8 Mb/mm^2 TCAM (Tsukamoto *et al*., 2015) during a 64 ms (refresh interval) is 3.072 µJ. This is 0.25% of 1.18 mJ (Micron, 2017) that DRAM spends for refresh operations. Additional energy is also consumed when searching and updating the counter table. The cost of this dynamic energy which is consumed when searching and updating the counter table is 15 pJ per activate command. Compared to the 13.89 nJ (Micron, 2017) that a DRAM consumes for ACT and PRE, 15 pJ is insignificant.

Sensitivity Study

By default, DEACT keeps a list of activated rows and moves highly activated (hot rows) to a row buffer. How many activations make a row hot is determined by the maximum activation permitted. The number of extra row buffers, the size of the activation counter, the maximum number of activation permitted, and the validation interval are the variables that impact the performance of DEACT. For each parameter, we use four sets of values as shown in Table 7. A total of 256 (4^4) experiments were conducted for sensitivity analysis; each experiment was run for 150 million simulation cycles.

The objective of this analysis is to study the effects of the aforementioned parameters on performance and energy consumption. Hence, we have implemented a simple counter table where old entities which are least activated are replaced with new entries when the table is full. In this study, we analyze the efficacy of DEACT by varying the size of the activation counter and expiration time of the list. We also analyze the effects of varying the activation thresh hold and the impact of extra row buffers on performance.

Activation Threshold

We have analyzed the impact of the activation threshold on performance on the 403.gcc workload of the CPU2006 benchmark. Figure 5 shows that a lower activation threshold yields a better hit rate.

Size of Activation Counter

The size of the activation counter should be large enough to track as many row activation as possible. A smaller size results in a record being overwritten by new records. On the other hand, a very big table may end up having unused space. Figure 7 shows the impact of varying the entry size of the counter table on performance. As can be seen in the figure, for the 403.gcc workload, 64 is the optimal value.

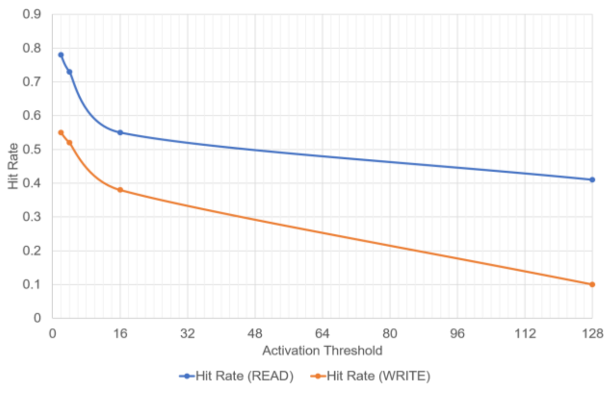

Fig. 5: Sensitivity analysis of the activation threshold on the performance of the 403.gcc workload

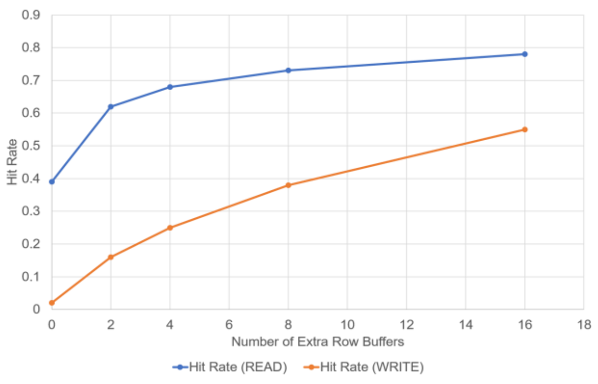

Fig. 6: Sensitivity analysis of the number of row buffers on the performance of the 403.gcc workload

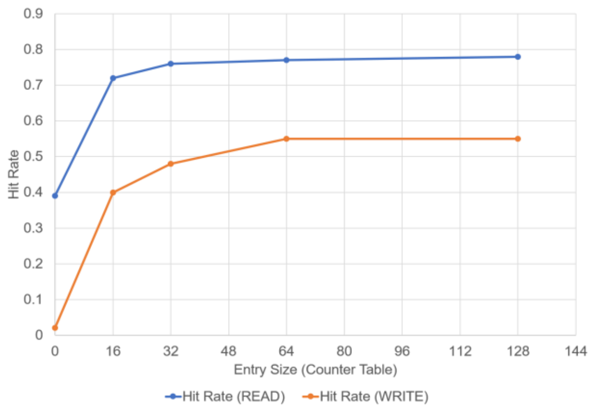

Fig. 7: Sensitivity analysis of the size of the counter table on the performance of the 403.gcc workload

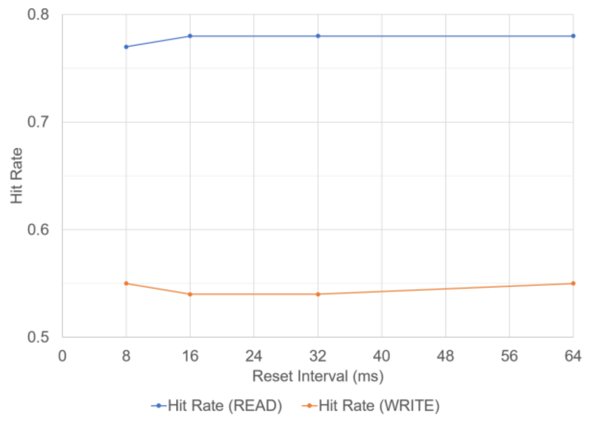

Fig. 8: Sensitivity analysis of the rest interval on the performance of the 403.gcc workload

Table 7: Parameters used in the sensitivity study of DEACT performance

Parameter	Case values
Activation threshold:	
Maximum number of activation a row should experience before it is declared hot and is moved to a row buffer	2, 4, 16, 128
Number of row buffers:	
Maximum number of extra row buffers per bank dedicated for keeping hot rows	2, 4, 8, 16
Size of activation counter:	
Number entries activation counter	16, 32, 64, 128
Reset interval:	
The time period that DEACT waits before resetting contents of the counters	8, 16, 32, 64

Reset Interval

Keeping old entries in the activation counter consumes space that would have been used by new entries. The counter table is cleared at a fixed interval. We test the effect of varying the validation interval on performance and the results of the study Fig. 8 show that the effect of this parameter is negligent. The reason could be attributed to the fact that frequently activated rows are always detected no matter what the reset window is. However, reducing the validation (reset) interval by a factor of x reduces the number of entries (space requirement) of the counter table by a factor of $2x/(x + 1)$.

Number of Row-Buffers

Theoretically, having many row buffers increases the hit rate. We analyzed the effects of additional two, four, eight, and sixteen-row buffers on performance. Figure 6 confirms that the hit rate increases with the number of row buffers.

Conclusion

Rowhammer is one of the big threats to computer security. Counter-based mitigation that detects excessive row activation, and tries to mitigate the effects of the Rowhammer problem by activating victim rows or by throttling DRAM operation. However, the associated performance and/or energy overhead of such implementations is significant.

We propose DEACT which solves all security vulnerabilities that are related to Rowhammer. Unlike existing mitigation, DEACT does not perform extra refreshes nor throttles any DRAM operation; it simply buffers a hot row in one of the row buffers dedicated for this purpose. DEACT is a counter based mitigation that keeps track of row activation at the RCD. We have shown the effect of using Four Activation Window (FAW) or Row Cycle (RC) timing parameters to estimate the size of the counter estimating the area overhead. FAW-based estimation reduces the storage overhead by a factor of 1.67 when compared to RC-based estimation.

DEACT not only eliminates Rowhammer, but it also performs better than standard DRAM. We tested DEACT using TPC and CPU-2006 benchmarks; it improves the hit rate on average by 41.16% for reads and 314.35% for writes for all workloads. The memory access latency has decreased by more than 18% for reads and 36% for writes on average. The queuing latency has also dropped by 30.7% for memory reads and 39.5% for memory writes.

Acknowledgment

We thank the publisher for allowing us to publish our research article. We would also like appreciate Mr. Jeffery Daniels, head of technology of Science Publications, for his relentless support during the publication process.

Funding Information

The authors have not received any financial support or funding to report.

Author’s Contributions

Tesfamichael Gebregziabher Gebrehiwot: Wrote the paper and analyzed the simulation results.

Fitsum Assamnew Andargie: Provided critical feedback and helped shape the article.

Mohammed Ismail: Supervised the work.

Ethics

This article is original and is not published elsewhere. The corresponding author confirms that all of the other authors have read and approved the manuscript.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this study.

References

Apple. (2015). About the security content of mac efi security update 2015-001. https://support.apple.com/en-us/HT204934

Aweke, Z. B., Yitbarek, S. F., Qiao, R., Das, R., Hicks, M., Oren, Y., & Austin, T. (2016). ANVIL: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices*, *51*(4), 743-755. https://doi.org/10.1145/2954679.2872390

Bains, K. S., & Halbert, J. B. (2016). Distributed row hammer tracking. *U.S. Patent No. 9,299,400B2*. Washington, DC: U.S. Patent and Trademark Office. https://patents.google.com/patent/US9299400B2/en

Bains, K. S., Halbert, J. B., Sah, S., & Greenfield, Z. (2015). *U.S. Patent No. 9,030,903*. Washington, DC: U.S. Patent and Trademark Office. https://patents.google.com/patent/US9030903B2/en

Bennett, T., Saroiu, S., Wolman, A., & Cojocar, L. (2021). Panopticon: A complete in-dram rowhammer mitigation. In *Workshop on DRAM Security (DRAMSec)*. https://alecw.azurewebsites.net/work/papers/dramsec-2021-panopticon.pdf

Bhattacharya, S., & Mukhopadhyay, D. (2016). Curious case of rowhammer: flipping secret exponent bits using timing analysis. In *Cryptographic Hardware and Embedded Systems-CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings 18*, (pp. 602-624). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-53140-2_29

Bock, C., Brasser, F., Gens, D., Liebchen, C., & Sadeghi, A. R. (2019, July). Rip-rh: Preventing rowhammer-based inter-process attacks. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, (pp. 561-572). https://doi.org/10.1145/3321705.3329827

Bosman, E., Razavi, K., Bos, H., & Giuffrida, C. (2016, May). Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE Symposium on Security and Privacy (SP)*, (pp. 987-1004). IEEE. https://ieeexplore.ieee.org/abstract/document/7546546

Brasser, F., Davi, L., Gens, D., Liebchen, C., & Sadeghi, A. R. (2017). CAn't touch this: Software-only mitigation against Rowhammer attacks targeting kernel memory. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, (pp. 117-130). https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-brasser.pdf

Chang, K. K., Nair, P. J., Lee, D., Ghose, S., Qureshi, M. K., & Mutlu, O. (2016, March). Low-cost inter-linked subarrays (LISA): Enabling fast inter-subarray data movement in DRAM. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, (pp. 568-580). IEEE. https://ieeexplore.ieee.org/abstract/document/7446095

Cheng, Y., Zhang, Z., Nepal, S., & Wang, Z. (2019). CATTmew: Defeating software-only physical kernel isolation. *IEEE Transactions on Dependable and Secure Computing*, *18*(4), 1989-2004. https://ieeexplore.ieee.org/abstract/document/8865632

Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, *55*(1), 58-75. https://doi.org/10.1016/j.jalgor.2003.12.001

de Ridder, F., Frigo, P., Vannacci, E., Bos, H., Giuffrida, C., & Razavi, K. (2021, August). SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript. In *USENIX Security Symposium,* (pp. 1001-1018). https://www.usenix.org/system/files/sec21fall-ridder.pdf

Devaux, F., & Ayrignac, R. (2021). *U.S. Patent No. 10,885,966*. Washington, DC: U.S. Patent and Trademark Office. https://patents.google.com/patent/US10885966B1/en

Fisch, D. E., & Plants, W. C. (2017). *U.S. Patent No. 9,812,185*. Washington, DC: U.S. Patent and Trademark Office. https://patents.googledd.com/patent/US9812185B2/en

Frigo, P., Giuffrida, C., Bos, H., & Razavi, K. (2018, May). Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In *2018 IEEE Symposium on Security and Privacy (sp),* (pp. 195-210). IEEE. https://ieeexplore.ieee.org/abstract/document/8418604

Frigo, P., Vannacc, E., Hassan, H., Van Der Veen, V., Mutlu, O., Giuffrida, C., ... & Razavi, K. (2020, May). TRRespass: Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy (SP)*, (pp. 747-762). IEEE. https://ieeexplore.ieee.org/abstract/document/9152631

Gans, D. D. (2021). Methods for Rowhammer mitigation and memory devices and systems employing the same. US Patent US20210280236A1.

Gautam, S. K., Kumar, A., & Manhas, S. K. (2018). Improvement of Rowhammering using metal nanoparticles in DRAM-A simulation study. *IEEE Electron Device Letters*, *39*(9), 1286-1289. https://ieeexplore.ieee.org/abstract/document/8423650

Gautam, S. K., Manhas, S. K., Kumar, A., & Pakala, M. (2020). Mitigating the passing word line induced soft errors in saddle fin DRAM. *IEEE Transactions on Electron Devices*, *67*(4), 1902-1905. https://ieeexplore.ieee.org/abstract/document/9025775

Gautam, S. K., Manhas, S. K., Kumar, A., Pakala, M., & Yieh, E. (2019). Rowhammering mitigation using metal nanowire in saddle fin DRAM. *IEEE Transactions on Electron Devices*, *66*(10), 4170-4175. https://ieeexplore.ieee.org/abstract/document/8798869

Gebrehiwot, T. G., Andargie, F. A., and Ismail, M. (2023). DDRSharp: A fast and extensible dram simulator. *Journal of Computer Science*, *19*(7): 836-846. https://thescipub.com/abstract/jcssp.2023.836.846

Gomez, H., Amaya, A., & Roa, E. (2016, November). DRAM row-hammer attack reduction using dummy cells. In *2016 IEEE Nordic Circuits and Systems Conference (NORCAS)*, (pp. 1-4). IEEE. https://ieeexplore.ieee.org/abstract/document/7792886

Google. (2014). Security: Nacl sandbox escape via dram “rowhammer” memory corruption. https://bugs.chromium.org/p/chromium/issues/detail?id=421090

Google. (2017). Rewrite non-temporal instructions. https://codereview.chromium.org/1269113003/

Greenfield, Z., & Tomer, L. E. V. Y. (2016). *U.S. Patent No. 9,251,885*. Washington, DC: U.S. Patent and Trademark Office. https://patents.google.com/patent/US9251885B2/en

Greenfield, Z., Bains, K. S., Schoenborn, T. Z., Mozak, C. P., & Halbert, J. B. (2015). *U.S. Patent No. 8,938,573*. Washington, DC: U.S. Patent and Trademark Office. https://patents.google.com/patent/US8938573B2/en

Greenfield, Z., Halbert, J. B., & Bains, K. S. (2014). *U.S. Patent Application No. 13/626,479*. https://patents.google.com/patent/US20140085995A1/en

Gruss, D. (2018). Software-based microarchitectural attacks. *it-Information Technology*, *60*(5-6), 335-341. https://doi.org/10.1515/itit-2018-0034

Gruss, D., Lipp, M., Schwarz, M., Genkin, D., Juffinger, J., O'Connell, S., ... & Yarom, Y. (2018, May). Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP),* (pp. 245-261). IEEE. https://ieeexplore.ieee.org/abstract/document/8418607

Gruss, D., Maurice, C., & Mangard, S. (2016). Rowhammer. js: A remote software-induced fault attack in javascript. In *Detection of Intrusions and Malware and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13,* (pp. 300-321). Springer International Publishing. https://doi.org/10.1007/978-3-319-40667-1_15

Hassan, H., Tugrul, Y. C., Kim, J. S., Van der Veen, V., Razavi, K., & Mutlu, O. (2021, October). Uncovering in-dram rowhammer protection mechanisms: A new methodology, custom rowhammer patterns and implications. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, (pp. 1198-1213). https://doi.org/10.1145/3466752.3480110

Henning, J. L. (2006). SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, *34*(4), 1-17. https://dl.acm.org/doi/pdf/10.1145/1186736.1186737

Hong, S., Frigo, P., Kaya, Y., Giuffrida, C., & Dumitras, T. (2019, August). Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *USENIX Security Symposium*, (pp. 497-514). https://www.usenix.org/system/files/sec19-hong.pdf

Hong, S., Kim, D., Lee, J., Oh, R., Yoo, C., Hwang, S., & Lee, J. (2023). DSAC: Low-Cost Rowhammer Mitigation Using In-DRAM Stochastic and Approximate Counting Algorithm. *arXiv preprint arXiv:2302.03591*. https://doi.org/10.48550/arXiv.2302.03591

HP. (2015). Hp moonshot component pack version 2015.05.0. http://h17007.www1.hp.com/us/en/enterprise/servers/products/ moonshot/component-pack/index.aspx

Irazoqui, G., Eisenbarth, T., & Sunar, B. (2018, March). Mascat: Preventing microarchitectural attacks before distribution. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, (pp. 377-388). https://doi.org/10.1145/3176258.3176316

Jang, Y., Lee, J., Lee, S., & Kim, T. (2017, October). SGX-Bomb: Locking down the processor via Rowhammer attack. In *Proceedings of the 2_{nd} Workshop on System Software for Trusted Execution*, (pp. 1-6). https://doi.org/10.1145/3152701.3152709

JEDEC. (2021). Jedec standard-jesd79-4d. 4d. https://www.jedec.org/sites/default/files/docs/JESD79-4D.pdf

Kim, D. H., Nair, P. J., & Qureshi, M. K. (2014a). Architectural support for mitigating Rowhammering in DRAM memories. *IEEE Computer Architecture Letters*, *14*(1), 9-12. https://ieeexplore.ieee.org/abstract/document/6840960

Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J. H., Lee, D., ... & Mutlu, O. (2014b). Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News*, *42*(3), 361-372. https://doi.org/10.1145/2678373.2665726

Kim, M. J., Park, J., Park, Y., Doh, W., Kim, N., Ham, T. J., ... & Ahn, J. H. (2022, April). Mithril: Cooperative Rowhammer protection on commodity dram leveraging managed refresh. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, (pp. 1156-1169). IEEE. https://ieeexplore.ieee.org/abstract/document/9773219

Kim, Y., Yang, W., & Mutlu, O. (2015). Ramulator: A fast and extensible DRAM simulator. *IEEE Computer Architecture Letters*, *15*(1), 45-49. https://ieeexplore.ieee.org/abstract/document/7063219

Konoth, R. K., Oliverio, M., Tatar, A. andriesse, D., Bos, H., Giuffrida, C., & Razavi, K. (2018). Zebram: Comprehensive and compatible software protection against rowhammer attacks. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, (pp. 697-710). https://www.usenix.org/system/files/osdi18-konoth.pdf

Kwong, A., Genkin, D., Gruss, D., & Yarom, Y. (2020, May). Rambleed: Reading bits in memory without accessing them. In *2020 IEEE Symposium on Security and Privacy (SP)*, (pp. 695-711). IEEE. https://ieeexplore.ieee.org/abstract/document/9152687

Lanteigne, M. (2016). How rowhammer could be used to exploit weaknesses in computer hardware. http://www.thirdio.com/rowhammer.pdf

Lee, E., Kang, I., Lee, S., Suh, G. E., & Ahn, J. H. (2019, June). TWiCe: Preventing row-hammering by exploiting time window counters. In *Proceedings of the 46th International Symposium on Computer Architecture*, (pp. 385-396). https://doi.org/10.1145/3307650.3322232

Lipp, M., Schwarz, M., Raab, L., Lamster, L., Aga, M. T., Maurice, C., & Gruss, D. (2020, September). Nethammer: Inducing rowhammer faults through network requests. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, (pp. 710-719). IEEE. https://ieeexplore.ieee.org/abstract/document/9229701

Liu, J., Jaiyen, B., Kim, Y., Wilkerson, C., & Mutlu, O. (2013). An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. *ACM SIGARCH Computer Architecture News*, *41*(3), 60-71. https://doi.org/10.1145/2508148.2485928

Liu, J., Jaiyen, B., Veras, R., & Mutlu, O. (2012). RAIDR: Retention-aware intelligent DRAM refresh. *ACM SIGARCH Computer Architecture News*, *40*(3), 1-12. https://doi.org/10.1145/2366231.2337161

Marazzi, M., Jattke, P., Solt, F., & Razavi, K. (2022, May). PROTRR: Principled yet optimal in-DRAM target row refresh. In *2022 IEEE Symposium on Security and Privacy (SP)*, (pp. 735-753). IEEE. https://ieeexplore.ieee.org/abstract/document/9833664

Metwally, A., Agrawal, D., & El Abbadi, A. (2005). Efficient computation of frequent and top-k elements in data streams. In *Database Theory-ICDT 2005: 10th International Conference, Edinburgh, UK, January 5-7, 2005. Proceedings 10*, (pp. 398-412). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-30570-5_27

Micron. (2015). 8gb: x4, x8, x16 DDR4 SDRAM features excessive row activation. https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/8gb_ddr4_sdram.pdf

Micron. (2017). System power calculators. https://www.micron.com/support/tools-and-utilities/power-calc

Misra, J., & Gries, D. (1982). Finding repeated elements. *Science of Computer Programming*, *2*(2), 143-152. https://doi.org/10.1016/0167-6423(82)90012-0

Mutlu, O. (2013, May). Memory scaling: A systems architecture perspective. In *2013 5th IEEE International Memory Workshop*, (pp. 21-25). IEEE. https://ieeexplore.ieee.org/abstract/document/6582088

Mutlu, O., Olgun, A., & Yağlıkcı, A. G. (2023, January). Fundamentally understanding and solving rowhammer. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, (pp. 461-468). https://doi.org/10.1145/3566097.3568350

Nair, P. J., Sridharan, V., & Qureshi, M. K. (2016). XED: Exposing on-die error detection information for strong memory reliability. *ACM SIGARCH Computer Architecture News*, *44*(3), 341-353. https://doi.org/10.1145/3007787.3001174

Park, Y., Kwon, W., Lee, E., Ham, T. J., Ahn, J. H., & Lee, J. W. (2020, October). Graphene: Strong yet lightweight Rowhammer protection. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, (pp. 1-13). IEEE. https://ieeexplore.ieee.org/abstract/document/9251863/

Rakin, A. S., Chowdhuryy, M. H. I., Yao, F., & Fan, D. (2022, May). Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE Symposium on Security and Privacy (SP),* (pp. 1157-1174). IEEE. https://ieeexplore.ieee.org/abstract/document/9833743

Razavi, K., Gras, B., Bosman, E., Preneel, B., Giuffrida, C., & Bos, H. (2016, August). Flip Feng Shui: Hammering a Needle in the Software Stack. In *USENIX Security symposium* (Vol. *25*, pp. 1-18). https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_razavi.pdf

Ryan, W., & Lin, S. (2009). *Channel codes: Classical and modern*. Cambridge university press. ISBN-10: 9781139483018.

Ryu, S. W., Min, K., Shin, J., Kwon, H., Nam, D., Oh, T., ... & Hong, S. (2017, December). Overcoming the reliability limitation in the ultimately scaled DRAM using silicon migration technique by hydrogen annealing. In *2017 IEEE International Electron Devices Meeting (IEDM)*, (pp. 21-6). IEEE. https://ieeexplore.ieee.org/abstract/document/8268437

Seaborn, M., & Dullien, T. (2015). Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat*, *15*, 71. https://www.cs.umd.edu/class/fall2019/cmsc818O/papers/rowhammer-kernel.pdf

Seyedzadeh, S. M., Jones, A. K., & Melhem, R. (2016). Counter-based tree structure for Rowhammering mitigation in DRAM. *IEEE Computer Architecture Letters*, *16*(1), 18-21.

Seyedzadeh, S. M., Jones, A. K., & Melhem, R. (2018, June). Mitigating wordline crosstalk using adaptive trees of counters. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, (pp. 612-623). IEEE. https://ieeexplore.ieee.org/abstract/document/8416859

Shutemov, K. A. (2015). Pagemap: Do not leak physical addresses to non-privileged userspace. https://lwn.net/Articles/642074/

Son, M., Park, H., Ahn, J., & Yoo, S. (2017, June). Making DRAM stronger against Rowhammering. In *Proceedings of the 54th Annual Design Automation Conference 2017*, (pp. 1-6). https://doi.org/10.1145/3061639.3062281

Taouil, M., Reinbrecht, C., Hamdioui, S., & Sepúlveda, J. (2021, July). LightRoAD: Lightweight Rowhammer Attack Detector. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, (pp. 362-367). IEEE. https://ieeexplore.ieee.org/abstract/document/9516766

Tatar, A., Konoth, R. K., Athanasopoulos, E., Giuffrida, C., Bos, H., & Razavi, K. (2018). Throwhammer: Rowhammer attacks over the network and defenses. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, (pp. 213-226). https://www.usenix.org/system/files/conference/atc18/atc18-tatar.pdf

Ting, D. (2018, July). Count-min: Optimal estimation and tight error bounds using empirical error distributions. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 2319-2328). https://doi.org/10.1145/3219819.3219975

Tobah, Y., Kwong, A., Kang, I., Genkin, D., & Shin, K. G. (2022, May). Spechammer: Combining spectre and rowhammer for new speculative attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, (pp. 681-698). IEEE. https://ieeexplore.ieee.org/abstract/document/9833802

TPC. (2023). TPC benchmark standards. https://www.tpc.org/

Tsukamoto, Y., Morimoto, M., Yabuuchi, M., Tanaka, M., & Nii, K. (2015, June). 1.8 Mbit/mm 2 ternary-CAM macro with 484 ps search access time in 16 nm Fin-FET bulk CMOS technology. In *2015 Symposium on VLSI Circuits (VLSI Circuits)*, (pp. C274-C275). IEEE. https://ieeexplore.ieee.org/abstract/document/7231286

Van Der Veen, V., Fratantonio, Y., Lindorfer, M., Gruss, D., Maurice, C., Vigna, G., ... & Giuffrida, C. (2016, October). Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, (pp. 1675-1689). https://doi.org/10.1145/2976749.2978406

Van der Veen, V., Lindorfer, M., Fratantonio, Y., Padmanabha Pillai, H., Vigna, G., Kruegel, C., ... & Razavi, K. (2018). GuardION: Practical mitigation of DMA-based rowhammer attacks on ARM. In *Detection of Intrusions and Malware and Vulnerability Assessment: 15th International Conference, DIMVA 2018, Saclay, France, June 28-29, 2018, Proceedings 15*, (pp. 92-113). Springer International Publishing. https://doi.org/10.1007/978-3-319-93411-2_5

Vig, S., Bhattacharya, S., Mukhopadhyay, D., & Lam, S. K. (2018, June). Rapid detection of Rowhammer attacks using dynamic skewed hash tree. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy* (pp. 1-8). https://doi.org/10.1145/3214292.3214299

Weissman, Z., Tiemann, T., Moghimi, D., Custodio, E., Eisenbarth, T., & Sunar, B. (2019). Jackhammer: Efficient Rowhammer on heterogeneous fpga-cpu platforms. *arXiv preprint arXiv:1912.11523*. https://doi.org/10.13154/tches.v2020.i3.169-195

Wu, X. C., Sherwood, T., Chong, F. T., & Li, Y. (2019, April). Protecting page tables from rowhammer attacks using monotonic pointers in dram true-cells. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, (pp. 645-657). https://doi.org/10.1145/3297858.3304039

Xiao, Y., Zhang, X., Zhang, Y., & Teodorescu, R. (2016, August). One Bit Flips, One Cloud Flops: Cross-VM Rowhammer Attacks and Privilege Escalation. In *USENIX Security Symposium*, (pp. 19-35). https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_xiao.pdf

Yağlıkçı, A. G., Kim, J. S., Devaux, F., & Mutlu, O. (2021). Security Analysis of the Silver Bullet Technique for RowHammer Prevention. *arXiv preprint arXiv:2106.07084*. https://doi.org/10.48550/arXiv.2106.07084

Yağlikçi, A. G., Patel, M., Kim, J. S., Azizi, R., Olgun, A., Orosa, L., ... & Mutlu, O. (2021, February). Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, (pp. 345-358). IEEE. https://ieeexplore.ieee.org/abstract/document/9407238

Yang, C. M., Wei, C. K., Chang, Y. J., Wu, T. C., Chen, H. P., & Lai, C. S. (2016). Suppression of Rowhammer effect by doping profile modification in saddle-fin array devices for sub-30-nm DRAM technology. *IEEE Transactions on Device and Materials Reliability*, *16*(4), 685-687. https://ieeexplore.ieee.org/abstract/document/7563821

Yao, F., Rakin, A. S., & Fan, D. (2020, August). Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. https://www.usenix.org/system/files/sec20-yao.pdf

Yokoyama, Y., Tanaka, M., Tanaka, K., Morimoto, M., Yabuuchi, M., Ishii, Y., & Tanaka, S. (2020, June). A 29.2 Mb/mm 2 Ultra High Density SRAM Macro using 7nm FinFET Technology with Dual-Edge Driven Wordline/Bitline and Write/Read-Assist Circuit. In *2020 IEEE Symposium on VLSI Circuits*, (pp. 1-2). IEEE. https://ieeexplore.ieee.org/abstract/document/9162985

You, J. M., & Yang, J. S. (2019, June). MRLoc: Mitigating Row-hammering based on memory Locality. In *Proceedings of the 56th Annual Design Automation Conference 2019*, (pp. 1-6). https://doi.org/10.1145/3316781.3317866

Zhang, Q., Pell, J., Canino-Koning, R., Howe, A. C., & Brown, C. T. (2014a). These are not the k-mers you are looking for: Efficient online k-mer counting using a probabilistic data structure. *PloS One*, *9*(7), e101271. https://doi.org/10.1371/journal.pone.0101271

Zhang, T., Chen, K., Xu, C., Sun, G., Wang, T., & Xie, Y. (2014b). Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation. *ACM SIGARCH Computer Architecture News*, *42*(3), 349-360. https://doi.org/10.1145/2678373.2665724

Zhang, Z., Cheng, Y., Liu, D., Nepal, S., Wang, Z., & Yarom, Y. (2020a, October). Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, (pp. 28-41). IEEE. https://ieeexplore.ieee.org/abstract/document/9251982

Zhang, Z., Zhan, Z., Balasubramanian, D., Li, B., Volgyesi, P., & Koutsoukos, X. (2020b, May). Leveraging em side-channel information to detect rowhammer attacks. In *2020 IEEE Symposium on Security and Privacy (SP)*, (pp. 729-746). IEEE. https://ieeexplore.ieee.org/abstract/document/9152769

Zhang, Z., Cheng, Y., Wang, M., He, W., Wang, W., Nepal, S., ... & Wu, C. (2022). {SoftTRR}: Protect Page Tables against Rowhammer Attacks using Software-only Target Row Refresh. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, (pp. 399-414). https://www.usenix.org/conference/atc22/presentation/zhang-zhi