Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı
I. E. Yüksel  A. Olgun  K. Kanellopoulos  Y. C. Tuğrul
A. G. Yağlıkçı  M. Sadrosadati  O. Mutlu

https://github.com/CMU-SAFARI/CoMeT
**Problem:** As DRAM becomes more vulnerable to read disturbance, existing RowHammer mitigation techniques either prevent bitflips (1) at low performance cost but with high area overheads or (2) at low area cost but with prohibitively large performance and energy overheads.

**Goal:** Prevent RowHammer bitflips with low area, performance, and energy overheads in highly RowHammer-vulnerable DRAM-based systems.

**Key Idea:** Use low-cost and scalable hash-based counters to accurately track DRAM rows.

**CoMeT:**
- tracks most DRAM rows with scalable hash-based counters by employing the Count-Min-Sketch technique to achieve a low area cost.
- tracks only a small set of DRAM rows that are activated many times with highly accurate per-DRAM-row activation counters to reduce performance penalties.

**Evaluation:** CoMeT achieves a good trade-off between area, performance and energy costs
- incurs significantly less area overhead (74.2×) compared to the state-of-the-art technique
- outperforms the state-of-the-art technique (by up to 39.1%)

[https://github.com/CMU-SAFARI/CoMeT](https://github.com/CMU-SAFARI/CoMeT)
Outline

Background and Problem

Goal and Key Idea

CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion
DRAM Organization

- DRAM Chip
- DRAM Bank
- DRAM Subarray

Chip I/O

Bank

Subarray

Bitline

DRAM Cell

Wordline

Row Buffer

DRAM Row
Repeatedly **opening** (activating) and **closing** (precharging) a DRAM row causes **RowHammer bitflips** in nearby cells.
The minimum number of activations that causes a bitflip is called **the RowHammer threshold**
Read Disturbance Vulnerabilities (II)

- DRAM chips are more vulnerable to read disturbance today.

- Read disturbance bitflips occur at much lower activation counts (more than two orders of magnitude decrease in less than a decade):

  \[ \text{HC}_{\text{first}} = \infty \] (all good)

  139K
  [Kim+, ISCA'14]

  9.6K
  [Kim+, ISCA'20]

  <1K
  [Luo+, ISCA'23]

Mitigation techniques against read disturbance attacks need to be **effective** and **efficient** for highly vulnerable systems.
Existing RowHammer Mitigations (I): Preventive Refresh

**DRAM Subarray**

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Victim Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 1</td>
<td>Victim Row</td>
</tr>
<tr>
<td><strong>Row 2</strong></td>
<td>Aggressor Row</td>
</tr>
<tr>
<td>Row 3</td>
<td>Victim Row</td>
</tr>
<tr>
<td>Row 4</td>
<td>Victim Row</td>
</tr>
</tbody>
</table>

**Refreshing** potential victim rows mitigates read disturbance bitflips
**Existing RowHammer Mitigations (II): Overview of Preventive Refresh-Based Mitigation Techniques**

**One ACT counter per DRAM row**

- **DRAM**: Row A, Row B, Row C, Row D
- **Processor Chip**: Counter A, Counter B, Counter C, Counter D

**One ACT counter per aggressor row**

- **Tag A**: Counter A, Counter B, Counter C
- **Tag B**: Counter B, Counter B, Counter C

**Shared Counters**

- **Tag A**: Counter A, Counter A
- **Tag B**: Counter B, Counter B

**Area Cost**

- **Very High**: Many DRAM rows (e.g., 128K per bank)
- **Very Low**: Few DRAM rows

**Performance & Energy Costs**

- **Very Low**: Low performance and energy costs
- **Low**: Moderate performance and energy costs
- **High**: High performance and energy costs

**High DRAM bandwidth consumption**

- **Low**: Low DRAM bandwidth consumption
No existing mitigation technique prevents RowHammer bitflips at low area, performance and energy costs.
Goal and Key Idea

CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion
Our Goal

Design a RowHammer mitigation technique that prevents RowHammer bitflips with low area, performance, and energy overheads in highly RowHammer-vulnerable DRAM-based systems.
Key Observation

Hash-based counters are low-cost:
(i) can be implemented with low-cost structures and
(ii) can aggregate many rows' activation counts together

Tag-based counters are highly accurate:
Each one tracks one row's activation count

**Diagram:**
1. **Hash Function (Example):** $H_0(ID) = ID \% 4$
2. **Counters:**
   
   **Mapping without tags:**
   - Row 0
   - Row 1
   - Row 2
   - Row 3
   - Row 4

   **Fixed number of counters:**
   - Counter 0
   - Counter 1
   - Counter 2
   - Counter 3

**Tags:**
- Row 0
- ...
Key Idea

1. Use low-cost and scalable hash-based counters to track most DRAM rows' activations with low area overhead.

2. Use highly accurate tag-based counters to track only a small set of DRAM rows to achieve low performance overhead.
Outline

Background and Problem

Goal and Key Idea

CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion
CoMeT Overview

Counter Table (CT):
- Maps each DRAM row to a group of low-cost hash-based counters as uniquely as possible by employing the Count-Min Sketch technique
- Triggers a preventive refresh to an aggressor row's victim rows when the aggressor's counter group reaches an activation threshold

Recent Aggressor Table (RAT):
- Allocates highly accurate per-DRAM-row counters for only a small set of DRAM rows that are activated many times

Reduces performance penalties by increasing tracking accuracy
Operation of CoMeT

1. Hash-based Counters
   - Counter Table-Base Estimation

2. Tag-based Counters
   - Recent Aggressor Table-Base Estimation
   - Used when there is a row ID match

3. COMPARE
   - Row A's ACT Count to $N_{PR}$
   - Preventive refresh threshold

   Preventively Refresh A's victim rows
Counter Table (CT): Count-Min-Sketch-based Row Tracking

- **Count-Min Sketch**: A hash-based frequent item counting technique

---

**Action Timeline**
- ACT A
- ACT A
- ACT B
- ACT C

**ESTIMATION**
- ACT_Count(A) = 3

**Hash Functions**
- \( H_0(A) \rightarrow 0 \)
- \( H_0(B) \rightarrow 4 \)
- \( H_0(C) \rightarrow 0 \)

**Counter Table**

- Counter 0: 3
- Counter 1: 0
- Counter 2: 0
- Counter 3: 0
- Counter 4: 1

**Counter Collision**
- Causes overestimation

**Actual ACT_Count(A) = 2**
Counter Table (CT): Count-Min-Sketch-based Row Tracking

- **Count-Min Sketch**: A hash-based frequent item counting technique

### Action Timeline

<table>
<thead>
<tr>
<th>ACT A</th>
<th>ACT A</th>
<th>ACT B</th>
<th>ACT C</th>
</tr>
</thead>
</table>

### Hash Functions

<table>
<thead>
<tr>
<th>H&lt;sub&gt;0&lt;/sub&gt;</th>
<th>H&lt;sub&gt;1&lt;/sub&gt;</th>
<th>H&lt;sub&gt;2&lt;/sub&gt;</th>
</tr>
</thead>
</table>

### Counters

<table>
<thead>
<tr>
<th>Column</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The minimum counter value is an upper bound for the actual activation count.
Counter Table (CT): Identifying Aggressor Rows

- CoMeT sets a *preventive refresh threshold* ($N_{PR}$) to timely refresh an aggressor row's victim rows to prevent bitflips.

### Action Timeline

1. **Identify** $A$ as an aggressor row
2. **Preventively** refresh $A$'s victim rows

### Hash Functions

- $H_0$
- $H_1$
- $H_2$

### Counter Table

- $N_{PR}$: Preventive Refresh Threshold
- **CounterGroup** $A$

### ESTIMATION

- $ACT\_Count(A) = N_{PR}$
- $ACT\_Count(C) = 0$

### Reset counters?

- **UNDERESTIMATION**
Counter Table (CT): Counter Saturation

- CoMeT **does not reset** any counter in CT after preventive refresh
- CT counters saturate at $N_{PR}$

**Action Timeline**

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT A</td>
<td>Preventively Refresh A's victim rows</td>
</tr>
<tr>
<td>ACT A</td>
<td>Preventively Refresh A's victim rows</td>
</tr>
</tbody>
</table>

**Hash Functions**

<table>
<thead>
<tr>
<th>Hash Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_0$</td>
</tr>
<tr>
<td>$H_1$</td>
</tr>
<tr>
<td>$H_2$</td>
</tr>
</tbody>
</table>

**Counter Table**

<table>
<thead>
<tr>
<th>Counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>$N_{PR}$</td>
</tr>
</tbody>
</table>

- $N_{PR}$: Preventive Refresh Threshold
- **CounterGroup** $A$

**ESTIMATION**

ACT_Count($A$) = $N_{PR}$

**OVERESTIMATION**

Preventively Refresh A's victim rows

**Unnecessary**: incurs performance and energy overheads

Actual ACT_Count($A$) after preventive refresh is 0
Recent Aggressor Table

- Allocates per-DRAM-row counters for **aggressor rows** to accurately estimate their activation counts **after preventive refreshes**
  - Implemented for **only a small set** of DRAM rows to maintain a low area cost

### Counter Table

**Hash Functions**
- $H_0$
- $H_1$
- $H_2$

**Counters**
- 0
- 1
- 2
- 3
- 4

![Counter Table Diagram]

* $N_{PR}$: Preventive Refresh Threshold

**Recent Aggressor Table**

- **Row Tag**: A
- **Counter**: 0

**TAG MATCH**

**ESTIMATION**

ACT_Count(A) = 0

If a DRAM row has a **Recent Aggressor Table entry**, CoMeT estimates its activation count **100% accurately**.
Operation of CoMeT

1. Hash-based Counters
2. Tag-based Counters

Counter Table
Recent Aggressor Table

Hash-based Counters - Counter Table-Based Estimation
Tag-based Counters - Recent Aggressor Table-Based Estimation

ACT_Count(A) == N_{PR}

Estimation used when there is a row ID match

Preventively Refresh A's victim rows

Preventive refresh threshold
Operation of CoMeT

CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı  İsmail Emir Yükseľ  Ataberk Olgun  Konstantinos Kanellopoulos
Yahya Can Tuğrul  A. Giray Yaşlıkçı  Mohammad Sadrosadati  Onur Mutlu
ETH Zürich

DRAM chips are increasingly more vulnerable to read-disturbance phenomena (e.g., RowHammer and RowPress), where repeatedly accessing DRAM rows causes bitflips in nearby rows due to DRAM density scaling. Under low RowHammer thresholds, existing RowHammer mitigations either incur high area overheads or degrade performance significantly.

We propose a new RowHammer mitigation mechanism, CoMeT, that prevents RowHammer bitflips with low area, performance, and energy costs in DRAM-based systems at very recent aggressor table.

1. Introduction

DRAM chips are susceptible to read-disturbance where repeatedly accessing a DRAM row (i.e., an aggressor row) can cause bitflips in physically nearby rows (i.e., victim rows) [1–13]. RowHammer is a type of read-disturbance phenomenon that is caused by repeatedly opening and closing (i.e., hammering) DRAM rows. Modern DRAM chips become more vulnerable to RowHammer as DRAM technology node size becomes smaller [1, 2, 4, 14–19]: the minimum number of row activations needed to cause a bitflip (i.e., RowHammer threshold

https://arxiv.org/abs/2402.18769

https://github.com/CMU-SAFARI/CoMeT
Outline

Background and Problem

Goal and Key Idea

CoMeT: Count-Min-Sketch-based Row Tracking

Evaluation

Conclusion
Evaluation Methodology

- **Performance and energy consumption evaluation**: cycle-level simulations using Ramulator [Kim+, CAL 2015] and DRAMPower [Chandrasekar+, DATE 2013]

- **System Configuration**:
  - **Processor**: 1 or 8 cores, 3.6GHz clock frequency, 4-wide issue, 128-entry instruction window
  - **DRAM**: DDR4, 1 channel, 2 rank/channel, 4 bank groups, 4 banks/bank group, 128K rows/bank
  - **Memory Ctrl.**: 64-entry read and write requests queues, Scheduling policy: FR-FCFS with a column cap of 16
    - **Last-Level Cache**: 8 MiB (single-core), 16 MiB (8-core)
  - **CoMeT**: *Counter Table*: 4 hash functions 512 counters per hash, *Recent Aggressor Table*: 128 entries

- **Comparison Points**: 4 state-of-the-art RowHammer mitigations
  - Graphene (best performing), Hydra (area-optimized best performing), Low Processor Chip Area Cost: REGA, PARA

- **Workloads**: 61 single-core applications and 56 8-core workload mixes
  - SPEC CPU2006, SPEC CPU2017, TPC, MediaBench, YCSB

[SAFARI](https://github.com/CMU-SAFARI/CoMeT)
Hardware Implementation

- Storage and area overhead analysis: CACTI
- Dual-rank area overhead comparison:

CoMeT stores **fewer bits** as the RowHammer threshold decreases.

CoMeT incurs a **significantly less area overhead** than Graphene and a **similar area overhead** to Hydra.
CoMeT prevents bitflips with very small average performance and DRAM energy overheads compared to a baseline system with no RowHammer mitigation.
Performance Comparison: Single-Core Applications

CoMeT incurs a small performance overhead (≤ 1.75%) over Graphene and outperforms Hydra (by up to 39.1%) at all RowHammer thresholds.

CoMeT outperforms all low-area-cost mitigations starting from the RowHammer threshold of 500.

1.75% (average)

1.75%
CoMeT maintains a performance overhead between Graphene's and Hydra's performance overheads
CoMeT incurs a small DRAM energy overhead (<1%) over Graphene and consumes less DRAM energy than Hydra.
CoMeT consumes less DRAM energy than all low-area-cost mitigations for all RowHammer thresholds.

CoMeT maintains a DRAM energy overhead between Graphene's and Hydra's DRAM energy overheads.
More in the Paper

• Security Analysis of CoMeT
• Sensitivity Analysis
  • Counter Table Configurations
  • Recent Aggressor Table Configurations
  • Counter Reset Period and Preventive Refresh Threshold values
• Performance Evaluation under Adversarial Workloads
• Performance Evaluation at High RowHammer Thresholds
• ...
More in the Paper

- Security Analysis of CoMeT
- Sensitivity Analysis
- Counter Table Configurations
- Recent Aggressor Table Configurations
- Counter Reset Period and Preventive Refresh Threshold values
- Performance Evaluation under Adversarial Workloads
- Comparison Against BlockHammer [Yaglikci +, HPCA'21]
- Performance Evaluation at High RowHammer Thresholds

CoMeT: Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı  İsmail Emir Yüksel  Ataberk Olgun  Konstantinos Kanellopoulos
Yahya Can Tuğrul  A. Giray Yağlıkçı  Mohammad Sadrosadati  Onur Mutlu

ETH Zürich

1. Introduction

DRAM chips are increasingly more vulnerable to read-disturbance phenomena (e.g., RowHammer and RowPress), where repeatedly accessing DRAM rows causes bitflips in nearby rows due to DRAM density scaling. Under low RowHammer thresholds, existing RowHammer mitigations either incur high area overheads or degrade performance significantly.

We propose a new RowHammer mitigation mechanism, CoMeT, that prevents RowHammer bitflips with low area, performance, and energy costs in DRAM-based systems at very...
CoMeT is Open Source and Artifact Evaluated

https://github.com/CMU-SAFARI/CoMeT
Outline

- Background and Problem
- Goal and Key Idea
- CoMeT: Count-Min-Sketch-based Row Tracking
- Evaluation
- Conclusion
Conclusion

CoMeT: a new Count-Min-Sketch-based Row Tracking Technique
- implements scalable hash-based counters by employing the Count-Min-Sketch technique to achieve a low area cost
- tracks only a small set of DRAM rows that are activated many times with highly accurate per-DRAM-row activation counters to reduce performance penalties

CoMeT achieves a good trade-off between area, performance and energy costs
- incurs significantly less area overhead (74.2×) compared to the state-of-the-art mitigation technique
- outperforms the state-of-the-art technique (by up to 39.1%)

https://github.com/CMU-SAFARI/CoMeT
CoMeT
Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

F. Nisa Bostancı
I. E. Yüksel  A. Olgun  K. Kanellopoulos  Y. C. Tuğrul
A. G. Yağlıkçı  M. Sadrosadati  O. Mutlu

https://github.com/CMU-SAFARI/CoMeT
CoMeT
Count-Min-Sketch-based Row Tracking to Mitigate RowHammer at Low Cost

BACKUP SLIDES

F. Nisa Bostancı
I. E. Yüksel  A. Olgun  K. Kanellopoulos  Y. C. Tuğrul
A. G. Yağlıkçı  M. Sadrosadati  O. Mutlu

https://github.com/CMU-SAFARI/CoMeT
Repeatedly opening (activating) and closing (precharging) a DRAM row causes RowHammer bitflips in nearby cells.

The minimum number of row activations needed to cause a bitflip (i.e., RowHammer threshold ($N_{RH}$)) has reduced by more than an order of magnitude in less than a decade.

RowPress is shown to lead to bitflips with one to two orders of magnitude fewer activations (than RowHammer) under realistic conditions.

[Kim+ ISCA'20] [Luo+ ISCA'23]
## Existing RowHammer Mitigations (II): Overview of Mitigation Techniques

<table>
<thead>
<tr>
<th>DRAM rows</th>
<th>CTRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Diagram: One ACT counter per DRAM row]</td>
<td>[Diagram: One ACT Counter per potential aggressor row]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tags</th>
<th>CTRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Diagram: &lt;1 counter per DRAM row]</td>
<td></td>
</tr>
</tbody>
</table>

### Performance Overhead
- **Very Low**
- **Low**
- **High**

### Area/Storage Overhead
- **Very High**
- **High**
- **Low**

- **~20 MiB** for \(2^{23}\) rows with 10-bit counters
- Increasing number of aggressor rows + expensive structures
- Low accuracy + Increasing usage of DRAM bandwidth
Accurately tracking DRAM row activations can be done by allocating per-row counters to potential aggressor rows.

As DRAM becomes more vulnerable to read disturbance, tracking all potential aggressor rows with tag-based counters results in a high area overhead.

At low $N_{RH}$, the number of potential aggressor rows increases significantly. A tag-based counter has a high area cost.

Accurate tracking enables low performance overhead by reducing unnecessary preventive refreshes.
As DRAM becomes more vulnerable to read disturbance, increased off-chip communication results in **high performance and energy overheads**.
Limitations of Existing Mitigations

No existing mitigation technique prevents RowHammer bitflips at low area, performance and energy costs.

High area cost due to many and expensive per-DRAM-row activation counters.

High performance and energy costs due to occupying the memory bandwidth with additional requests.
Operation of CoMeT

ACT A

Counter Table

Hash Functions

H0

H1

H2

Counters

0 1 2 3 4

CounterGroupA

CoMeT

Row Tag

Counter

TAG MATCH

Recent Aggressor Table

CT ESTIMATION
ACT_Count(A) = MIN(CounterGroupA)

RAT ESTIMATION
ACT_Count(A) = RAT.Counter(A)

used when there is a tag match

ESTIMATION
ACT_Count(A) == N_{PR}

Preventively Refresh A's victim rows

SAFARI
CoMeT Overview

1. **Counter Table**
   - $CG(X)$:
     - $H_0(X)=0 \rightarrow C_{00}$
     - $H_1(X)=2 \rightarrow C_{12}$
   - Table:
     - $H_0$, $H_1$, $H_2$, ..., $H_{k-1}$
     - Rows: $C_{00}$, $C_{12}$, ...

2. **Recent Aggressor Table**
   - **DRAM Row Tag**
     - **Counter**
   - **Row ID X**
   - **N_rAT Entries**
   - **RAT_Ctr**

3. **Comparator**

4. **Min_Ctr**

5. **Tag Match**

6. **SELECT**
   - **Num_ACT**
   - $\geq$ **N_PR**

7. **Y: Preventive Refresh**

---

**SAFARI**

47
Early Preventive Refresh at Coarse Granularity

**Hash Functions**

<table>
<thead>
<tr>
<th>Hash Function</th>
<th>Counter Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_0$</td>
<td>$N_{PR}$</td>
</tr>
<tr>
<td>$H_1$</td>
<td>$N_{PR}$</td>
</tr>
</tbody>
</table>

**Counters**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{PR}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Row Tag**

- A

**Counter**

... 

**Recent Aggressor Table**

Entries can be evicted if there are many aggressor rows.

CoMeT falls back to Counter Table estimation ($N_{PR}$).

Preventively refreshes victim rows.

SAFARI
Configuring CoMeT – Sensitivity Analysis: Counter Table and Recent Aggressor Table

**Configuring Counter Table**

- **Counters per Hash Function (N\textsubscript{Counters})**
  - 128
  - 256
  - 512
  - 1024
  - 2048

**Configuring RAT**

- **Number of RAT Entries (N\textsubscript{RAT\_Entries})**
  - 32
  - 64
  - 128
  - 256
  - 512
Configuring CoMeT – Sensitivity Analysis: Counter Table and Recent Aggressor Table

CoMeT configuration that achieves both performance and area efficiency:
- **Counter Table** with **4 hash functions** and **512 counters per hash function**
- **Recent Aggressor Table** with **128 entry**

(more analyses in the paper)
Configuring CoMeT – Sensitivity Analysis: Counter Reset Period and $N_{PR}$

\[ \text{Counter Refresh Period} = \frac{t_{REFW}}{k} \]  \hspace{1cm} (1)

\[ N_{PR} = \frac{N_{RH}}{k + 1} \]  \hspace{1cm} (2)
The effect of EPRT and RAT Miss History Length on Performance and DRAM energy consumption.
### Hardware Implementation

- **Storage and area overhead analysis:** CACTI
  - Logic circuitry overhead: Verilog HDL implementation and Synopsys DC

- **Dual-rank area overhead comparison:**

<table>
<thead>
<tr>
<th></th>
<th>(N_{RH}=1K)</th>
<th>(N_{RH}=500)</th>
<th>(N_{RH}=250)</th>
<th>(N_{RH}=125)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KB</td>
<td>(mm^2)</td>
<td>KB</td>
<td>(mm^2)</td>
</tr>
<tr>
<td>CoMeT</td>
<td>76.5</td>
<td>0.09</td>
<td>68.0</td>
<td>0.08</td>
</tr>
<tr>
<td>CT (SRAM)</td>
<td>64.0</td>
<td>0.05</td>
<td>56.0</td>
<td>0.05</td>
</tr>
<tr>
<td>RAT (CAM)</td>
<td>12.5</td>
<td>0.03</td>
<td>12.0</td>
<td>0.03</td>
</tr>
<tr>
<td>Logic Circuitry</td>
<td>-</td>
<td>0.005</td>
<td>-</td>
<td>0.005</td>
</tr>
<tr>
<td>Graphene [86]</td>
<td>207.2</td>
<td>0.49</td>
<td>398.4</td>
<td>1.13</td>
</tr>
<tr>
<td>Hydra [90]</td>
<td>61.6</td>
<td>0.08</td>
<td>56.5</td>
<td>0.08</td>
</tr>
</tbody>
</table>

As \(N_{RH}\) decreases, CoMeT's area and storage overheads **decrease** due to storing **fewer bits** for its counters.
Hardware Implementation

• Storage and area overhead analysis: CACTI
  • Logic circuitry overhead: Verilog HDL implementation and Synopsys DC

• Dual-rank area overhead comparison:

<table>
<thead>
<tr>
<th></th>
<th>$N_{RH}=1K$</th>
<th>$N_{RH}=500$</th>
<th>$N_{RH}=250$</th>
<th>$N_{RH}=125$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KB</td>
<td>$mm^2$</td>
<td>KB</td>
<td>$mm^2$</td>
</tr>
<tr>
<td>CoMeT</td>
<td>76.5</td>
<td>0.09</td>
<td>68.0</td>
<td>0.08</td>
</tr>
<tr>
<td>CT (SRAM)</td>
<td>64.0</td>
<td>0.05</td>
<td>56.0</td>
<td>0.05</td>
</tr>
<tr>
<td>RAT (CAM)</td>
<td>12.5</td>
<td>0.03</td>
<td>12.0</td>
<td>0.03</td>
</tr>
<tr>
<td>Logic Circuitry</td>
<td>-</td>
<td>0.005</td>
<td>-</td>
<td>0.005</td>
</tr>
<tr>
<td>Graphene [86]</td>
<td>207.2</td>
<td>0.49</td>
<td>398.4</td>
<td>1.13</td>
</tr>
<tr>
<td>Hydra [90]$^8$</td>
<td>61.6</td>
<td>0.08</td>
<td>56.5</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Compared to the best performing state-of-the-art mitigation, CoMeT induces significantly less area overhead. ()
Hardware Implementation

- **Storage and area overhead analysis:** CACTI
  - Logic circuitry overhead: Verilog HDL implementation and Synopsys DC

- **Dual-rank area overhead comparison:**

<table>
<thead>
<tr>
<th></th>
<th>$N_{RH}=1K$</th>
<th></th>
<th>$N_{RH}=500$</th>
<th></th>
<th>$N_{RH}=250$</th>
<th></th>
<th>$N_{RH}=125$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KB</td>
<td>$mm^2$</td>
<td>KB</td>
<td>$mm^2$</td>
<td>KB</td>
<td>$mm^2$</td>
<td>KB</td>
</tr>
<tr>
<td>CoMeT</td>
<td>76.5</td>
<td>0.09</td>
<td>68.0</td>
<td>0.08</td>
<td>59.5</td>
<td>0.07</td>
<td>51.0</td>
</tr>
<tr>
<td>CT (SRAM)</td>
<td>64.0</td>
<td>0.05</td>
<td>56.0</td>
<td>0.05</td>
<td>48.0</td>
<td>0.04</td>
<td>40.0</td>
</tr>
<tr>
<td>RAT (CAM)</td>
<td>12.5</td>
<td>0.03</td>
<td>12.0</td>
<td>0.03</td>
<td>11.5</td>
<td>0.03</td>
<td>11.0</td>
</tr>
<tr>
<td>Logic Circuitry</td>
<td>-</td>
<td>0.005</td>
<td>-</td>
<td>0.005</td>
<td>-</td>
<td>0.005</td>
<td>-</td>
</tr>
<tr>
<td>Graphene [86]</td>
<td>207.2</td>
<td>0.49</td>
<td>398.4</td>
<td>1.13</td>
<td>765.0</td>
<td>3.01</td>
<td>1466.2</td>
</tr>
<tr>
<td>Hydra [90]</td>
<td>61.6</td>
<td>0.08</td>
<td>56.5</td>
<td>0.08</td>
<td>51.4</td>
<td>0.07</td>
<td>46.8</td>
</tr>
</tbody>
</table>

Compared to the best performing low-area-cost mitigation, CoMeT induces similar area overhead
Single-Core Performance Workload Breakdown (Medium and High Intensity)
Single-Core DRAM Energy Workload Breakdown (Medium and High Intensity)
CoMeT incurs negligible additional performance overhead on benign workloads when a traditional RowHammer attack is running at the same time.
Comparison Against BlockHammer (I)
Tracker Comparison

- CoMeT and BlockHammer employ different algorithms and this results in different DRAM-row-to-counter mappings.

Average # of unique rows that are activated at least 125 times by benign workloads

the average number of unique rows touched at least once by benign workloads

When tracking at most 2,500 unique rows, CoMeT’s tracker outperforms BlockHammer.
CoMeT outperforms BlockHammer due to BlockHammer's (i) high false positive rate and (ii) increased memory request latencies due to throttling.
Performance Comparison: 8-Core Workloads

CoMeT's performance overhead over Graphene is **0.9%** and **14.9%** at $N_{RH} = 1K$ and 125, respectively.

CoMeT **outperforms** Hydra for all RowHammer thresholds (by up to **3.2×** and **11.9%** on average at $N_{RH} = 125$).

CoMeT **outperforms** all low-cost RowHammer mitigations starting from $N_{RH}=250$. 
Performance Comparison: 8-Core Workloads

CoMeT maintains a performance overhead between Graphene's and Hydra's performance overheads.

CoMeT outperforms all low-area-cost mitigations starting from the RowHammer threshold of 250.

More preventive refreshes
Single-Core Comparison – Radar Chart
Multi-Core Comparison – Radar Chart