Hardware Prefetcher-based Rowhammer Attack

Yashwant Kumar Balivada, Maccoy Merrell, Stavros Kalafatis, Paul V. Gratz

Department of Electrical and Computer Engineering, Texas A&M University

Repeated DRAM row activations can lead to disturbance-based faults such as those exploited by Rowhammer in DDR3, DDR4 and DDR5 systems. While prior work has focused on software-generated hammering patterns, modern processors employ hardware prefetchers that speculatively generate memory requests capable of increasing DRAM activation activity. In this work, we investigate hardware prefetchers as a potential source of indirect row activations and demonstrate that they can be trained to generate Rowhammer-style access patterns across DRAM banks. We evaluate these behaviors using cycle-level simulation and extend our study toward real-system validation by integrating a Signature Path Prefetcher (SPP) into a BOOMv2-based processor within the Chipyard framework. Our results highlight the need to consider speculative prefetch activity within the threat model of DRAM disturbance-based faults.

1. Introduction

RowHammer is a disturbance-based memory phenomenon in which repeated activations of DRAM rows induce electrical interference that can corrupt data stored in neighboring rows [1–4]. When an aggressor row is activated repeatedly within a refresh interval, charge leakage can alter the stored values of adjacent victim rows, resulting in unintended bit flips. Experimental studies have demonstrated this behavior across multiple DRAM generations, including DDR3, DDR4, and DDR5 [5,6], establishing RowHammer as a persistent challenge for both memory reliability and security. Disturbance-induced bit flips can be exploited to compromise system integrity, enabling attacks such as privilege escalation, data corruption, and denial of service [4, 7]. Traditional RowHammer attacks rely on carefully crafted binaries that repeatedly access selected memory locations while forcing cache evictions to maximize DRAM row activations. Implicit in this threat model is the assumption that DRAM activity is largely driven by software-visible memory accesses and can therefore be characterized through instruction streams, memory traces, or performance-counter observations. Modern processors increasingly violate this assumption through the use of hardware prefetchers. As illustrated in Fig. 1, prefetchers proactively generate memory requests beyond those explicitly issued by software by learning and predicting future access patterns. These speculative requests traverse the cache hierarchy and may ultimately reach DRAM, creating row activations that are not directly attributable to program instructions. Hardware prefetchers possess several properties that make them particularly relevant in the context of RowHammer. First, many prefetchers are trainable, allowing carefully constructed access patterns to influence their prediction logic and induce controlled sequences of speculative memory requests. Second,

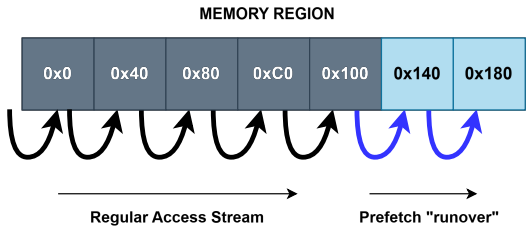

Figure 1: Hardware Prefetcher which speculatively runs ahead to prefetch data, causing more DRAM activations.

prefetch-generated accesses occur entirely within the microarchitecture and are often invisible to software-level monitoring mechanisms. Third, prefetchers can amplify DRAM activity by generating additional requests beyond those demanded by the application. Consequently, DRAM row activations can become partially decoupled from the program's visible memory behavior, making disturbance-inducing activity difficult to infer through conventional software-based analysis. This observation has important security implications. Many deployed systems rely on software-visible indicators, such as instruction streams, memory-access traces, or performance-counter activity, to reason about memory behavior and potential RowHammer threats. However, these approaches implicitly assume that DRAM activations closely track explicit load and store operations. Hardware prefetchers violate this assumption by autonomously generating memory requests that can significantly increase row activations while remaining largely hidden from software observation. This disconnect between observable program behavior and underlying DRAM activity motivates the need to reevaluate RowHammer threat models in the presence of autonomous microarchitectural memory-generation mechanisms.

These characteristics suggest that Rowhammer activity may be more covert than previously assumed. Even when software access patterns appear benign, prefetch-generated memory requests may continue to access deeper levels of the memory hierarchy, generating additional row activations without explicit software intent. Because these speculative references originate within the cache hierarchy and propagate toward DRAM, they may bypass software-based detection mechanisms.

This paper offers the following contributions:

- We investigate the role of hardware prefetchers as a potential mechanism for generating indirect Rowhammer activity.
- We demonstrate that prefetchers can be intentionally trained to produce memory access patterns that resemble Rowhammer-style hammering across DRAM banks.
- We evaluate these behaviors using cycle-level simulation using Signature Path lookahead Prefetcher(SPP) [8].
- We extend our study toward real-system validation by inte-

grating the SPP into a BOOMv2-based processor [9].
- Our findings highlight the need to consider speculative prefetch activity when analyzing Rowhammer threats and designing future mitigation mechanisms.

2. Background

2.1. Prior Works

Recent research has demonstrated that RowHammer can be triggered through a wide range of microarchitectural mechanisms beyond explicit load and store instructions. These studies collectively challenge the traditional assumption that RowHammer attacks require software-visible, high-frequency memory accesses, showing instead that implicit hardware activity can generate sufficient DRAM row activations to induce bit flips. RhoHammer [10] showed that software-controlled prefetch instructions, such as x86 *PREFETCH* and ARM *PRFM*, can generate frequent DRAM row activations capable of inducing RowHammer bit flips. Because prefetch instructions are non-blocking and often bypass instruction-retirement accounting and performance-counter-based monitoring, they provide a stealthy hammering primitive that evades many existing detection mechanisms. Similarly, PTHammer [11] demonstrated that implicit memory accesses arising from page-table walks across the user–kernel boundary can be exploited to induce RowHammer effects without requiring direct high-frequency memory accesses from user space. Moving beyond software-controlled mechanisms, GhostKnight [12] established that speculative execution itself can generate DRAM activation patterns sufficient to trigger RowHammer on DDR3 systems, while SpecHammer [13] further explored the interaction between transient execution and RowHammer, revealing a bidirectional relationship in which RowHammer-induced faults can amplify speculative execution attacks and vice versa. At the system level, MOESI-Prime [14] showed that coherence-induced traffic in ccNUMA systems, including speculative coherence reads and redundant directory updates, can inherently generate enough DRAM activity to trigger RowHammer under otherwise benign workloads. Taken together, these works reveal a broader trend: RowHammer vulnerabilities increasingly arise from implicit microarchitectural behavior rather than solely from adversarial memory-access streams generated by software. However, existing studies primarily focus on software-invoked prefetch instructions, page-table activity, speculative execution, or coherence protocols. In contrast, we investigate hardware prefetchers as an autonomous source of DRAM row activations. Unlike software-controlled mechanisms, hardware prefetchers generate memory requests entirely within the microarchitecture based on dynamically learned access patterns. Consequently, they can amplify row activations without requiring a corresponding increase in demand accesses from the core, effectively decoupling DRAM activity from the program's visible memory behavior. This separation significantly reduces the effectiveness of software-level monitoring approaches based on instruction tracing, retired memory operations, or performance counters, thereby exposing a largely unexplored and highly stealthy RowHammer attack surface. Our evaluation further considers a multi-bank hammering setting motivated by recent trends in RowHammer attack design. Prior work has shown that exploiting bank-level parallelism is an effective strategy for maximizing the number of DRAM activations within a refresh interval. For example, SledgeHammer [15] distributes hammering activity across multiple banks to increase activation throughput and improve attack effectiveness. Likewise, ABACuS [16] studies RowHammer behavior under prefetch-like access patterns, including next-line, next-8, and next-32 accesses, and observes that such patterns can activate rows with identical row indices across multiple banks before revisiting a previously activated row. While ABACuS leverages this observation to design scalable all-bank activation-counting defenses, it does not characterize hardware prefetchers as an autonomous source of RowHammer-inducing DRAM activations. Our work complements ABACuS by demonstrating that real hardware prefetchers can generate and amplify row activations in a manner that creates a practical attack surface. Building on these insights, our bank-mirroring attack exploits hardware prefetchers to generate coordinated activations across multiple banks, thereby increasing activation parallelism and accelerating RowHammer effects. This setting provides a natural framework for evaluating the security implications of autonomous prefetcher-induced DRAM activity and demonstrates how hardware prefetchers can serve as powerful activation amplifiers in modern memory systems.

2.2. Rowhammer Attacks

RowHammer (RH) attacks targeting in-memory binaries typically follow a structured workflow to induce controlled memory corruption. The attack begins by analyzing program memory layouts to identify candidate physical locations where bit flips can reliably impact critical data or instructions. Once such locations are identified, the attacker performs targeted row activation patterns to induce disturbance errors in DRAM, eventually causing bit flips in adjacent victim rows. These corrupted values can then be exploited to modify code or data in memory, enabling attacks such as privilege escalation or control-flow manipulation [4, 17]. As shown in Fig. 2, modern systems such as AMD Zen4 employ complex physical address-to-DRAM mappings that interleave addresses across channels, ranks, banks, bank groups, and rows [6]. Effective RowHammer attacks therefore often rely on understanding or inferring this mapping to precisely steer memory accesses toward targeted DRAM structures. By exploiting knowledge of how physical address bits map to bank and row indices, an attacker can construct access patterns that concentrate activations on selected banks and maximize disturbance in nearby rows. However, generating such carefully structured access patterns is non-trivial in practice. DRAM address mappings scramble contiguous physical addresses across multiple DRAM dimensions, causing effective hammering patterns to appear irregular in physical address space. While simple prefetching mechanisms (e.g., stride-based prefetchers) can capture regular access patterns, they are generally insufficient for modeling the com-

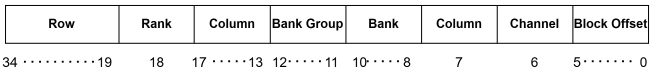

Figure 2: Zen4 address mapping extrapolated by reverse-engineering

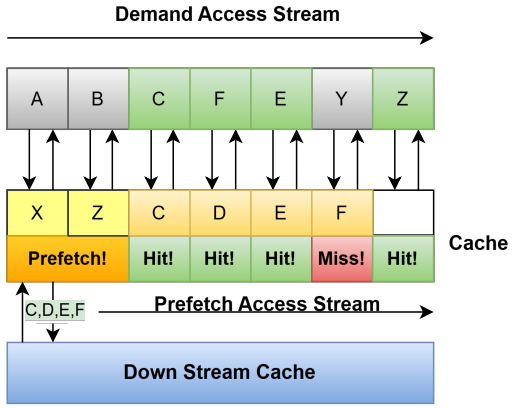

Figure 3: Hardware Prefetching

plex, multi-dimensional access sequences required for effective RowHammer amplification. As a result, practical attacks typically rely on explicitly crafted memory access sequences that directly encode bank- and row-level targeting rather than depending on hardware prefetchers to infer such patterns. With knowledge of the address mapping, an attacker can further refine access patterns to manipulate bank and channel selection bits, thereby steering repeated accesses toward specific DRAM banks. Similarly, controlling row index bits enables systematic traversal and repeated activation of targeted rows within a bank, increasing the likelihood of inducing disturbance errors in a controlled and repeatable manner.

Despite the existence of Rowhammer mitigations, many older deployed systems either lack protections or rely on mechanisms known to be bypassable. In particular, defenses such as Target Row Refresh (TRR) [18] have been shown to provide incomplete coverage, as their internal tracking mechanisms monitor only a limited number of frequently activated rows. This leaves them vulnerable to adversarial access patterns that distribute activations across multiple rows, enabling bit flips even in systems with mitigation support.

2.3. Prefetching

Hardware prefetchers are widely used in modern processors to improve performance by speculatively fetching cache lines before they are explicitly requested by the processor, as shown in Fig. 3. By predicting future memory accesses, they reduce effective memory latency and increase memory-level parallelism. However, these speculative requests introduce additional memory traffic that is not directly initiated by software, thereby increasing both the volume and rate of accesses issued to the DRAM subsystem. While accurate prefetching improves performance, inaccurate or overly aggressive prefetches can lead to cache pollution, redundant memory transfers, and cache thrashing. More importantly, prefetchers may generate memory accesses to locations that are never explicitly referenced by the executing program, thereby inducing additional DRAM row activations that are not reflected in the architectural instruction stream. In conjunction with DRAM address mapping, which distributes physical addresses across channels, ranks, banks, and rows, even spatially or stride-based prefetchers can produce structured access streams that traverse multiple rows and banks. This can systematically increase row activation frequency, particularly when predicted streams cross row boundaries within the same bank. Because prefetch requests are generated internally within the processor pipeline, they are largely invisible to software-level monitoring mechanisms, making it difficult to attribute resulting memory traffic to program behavior. As a result, prefetch-induced memory activity can introduce additional and often hidden row activations, highlighting the need to consider speculative memory generation when analyzing DRAM disturbance effects. The effectiveness of prefetch-assisted RowHammer amplification, however, depends on the underlying prefetcher design. In particular, prefetchers differ in whether they operate on virtual or physical addresses. Virtual-address (VA)-based prefetchers can, in principle, be trained to generate requests that span page boundaries, potentially increasing the scope of induced memory activity. In contrast, this work focuses on a physical-address (PA)-based prefetcher (SPP), where predictions are generated in physical address space. Due to this design choice, SPP primarily learns page-local patterns, and a single triggering stream does not generally induce prefetches that cross page boundaries. This restriction shapes the observed prefetch-driven access patterns and bounds the extent of cross-page activation amplification considered in this study.

3. Prefetcher-induced Rowhammer Attacks

Hardware prefetchers can significantly increase memory traffic by issuing speculative requests that are not directly triggered by program instructions, effectively generating bursts of memory accesses that may traverse large regions of memory. These speculative accesses translate into additional DRAM traffic, where each cache miss may trigger row activations within the memory banks. Although such activations occur as an unintended side effect of performance optimization, repeated and aggressive prefetching can substantially increase the rate and spatial distribution of DRAM row activations. This observation raises an important question: if speculative prefetch activity can generate frequent activations incidentally, then carefully influencing access patterns may allow these activations to be induced in a more controlled manner. Consequently, prefetchers have the potential to amplify activation intensity and unintentionally create conditions favorable for disturbance-based effects, including Rowhammer attacks.

Without loss of generality, in this work we first demonstrate prefetcher-based attacks that leverage the Signature Path Prefetcher (SPP), a well known and commonly used prefetcher in academic [8, 19–22] and commercial [23] works. To that end, we first discuss how SPP works and then discuss how it can be leveraged to create a Rowhammer attack. In Section 4 we show that other prefetchers are susceptible to this form of

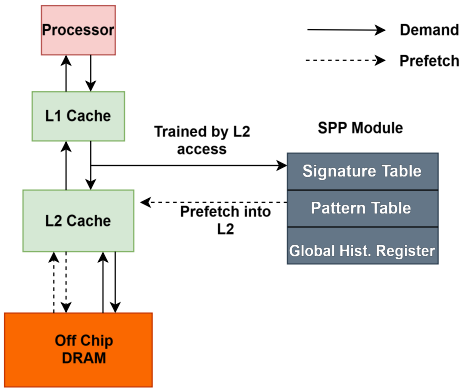

Figure 4: SPP Architecture

attack as well.

3.1. Signature Path Prefetcher Microarchitecture

SPP is a correlation-based hardware prefetching mechanism that predicts future memory accesses by tracking the history of access patterns. SPP generates a compact signature representing the recent sequence of memory accesses and uses this signature to index pattern tables that store likely future offsets. By associating signatures with previously observed memory transitions, SPP is able to capture complex access patterns beyond simple stride-based behavior.

As shown in Fig. 4 SPP consists of three key components: the Signature Table (ST), Pattern History Table (PHT), and Global History Register (GHR). The ST is indexed using the page number and performs a tag comparison to identify the corresponding page entry, after which the stored signature is retrieved and used to index the PHT. Each PHT entry stores candidate deltas along with confidence values, and the delta with the highest confidence is selected to generate a prefetch. The selected delta is used to update the signature, enabling recursive indexing of the PHT until the confidence of the predicted delta falls below a predefined threshold.

The GHR is used to track page boundary crossings, allowing SPP to maintain prediction continuity without explicitly tracking new pages. SPP also monitors the number of issued and useful prefetches to dynamically throttle prefetch generation when accuracy decreases. Additionally, a prefetch filter integrated within the ST suppresses redundant prefetch requests, reducing unnecessary memory traffic.

While SPP improves memory performance by increasing prefetch coverage and reducing effective memory latency, its speculative nature introduces additional memory requests that may increase DRAM row activations. These internally generated accesses can occur even when the corresponding memory locations are not explicitly referenced by the program, potentially altering memory access characteristics and increasing activation activity within the DRAM subsystem.

3.2. SPP's Bank Mirroring Attack

We leverage SPP to orchestrate a bank-parallelized Rowhammer attack, similar in spirit to prior multi-bank attacks, but critically differs in that the memory access stream is amplified

Algorithm 1: SPP-Based Bank Mirroring Attack

Input: `arr`, `arr1`, `arr2` of size N

```
1  stride = 4;
   // Initial training phase
2  for i ← 0 to TRAIN_PAGE do
3      int x = arr[i];
4  wait(3);
5  uint64_t idx = 0;
6  while true do
       // Access arr1
7      int x = arr1[idx % N];
       // Prefetch triggers
8      wait(1);
       // Access arr2
9      int y = arr2[(idx + 1) % N];
       // Prefetch triggers
10     wait(1);
11     idx += stride;
12     if idx >= N then
13         idx = 0;
```

by the hardware prefetcher rather than being entirely driven by the attacking process (software). By exploiting bank-level parallelism, the attack activates aggressor rows across multiple banks simultaneously with minimal degradation compared to single-bank hammering. Prefetchers are naturally capable of generating cross-bank memory requests. This property enables the attacker to indirectly steer accesses across multiple banks by carefully constructing access patterns that the prefetcher learns and amplifies, thereby extending the attack surface beyond what is explicitly issued by the CPU.

In our SPP-based bank mirroring attack, we allocate multiple memory regions such that each array resides on a separate 4KB page, enabling controlled placement across DRAM rows. To ensure that these pages are mapped onto specific and alternate DRAM rows, we leverage memory waylaying [24], which helps steer page placement toward desired physical locations without relying on explicit physical address control.

As illustrated in Fig. 5, the attack operates by alternately accessing two pages (e.g., Page 1 and Page 2), which are mapped to different DRAM rows. An initial training phase issues repeated accesses to a training page, enabling the hardware prefetcher to learn a stable stride-based pattern. Once this behavior is established, accesses to the target pages follow a similar structure, allowing SPP to recognize the pattern and begin generating speculative prefetch requests. For inducing the disturbance we alternate the accesses to both the pages as shown by *solid black* arrows (demand accesses issued by the program), and *blue dotted* arrows (prefetch requests by SPP).

Algorithm 1 details this process step-by-step, where repeated accesses are issued to a training page (step 1-4) to allow the hardware prefetcher to learn and stabilize a stride-based access pattern. Once the prefetcher is sufficiently trained, the attack transitions to the exploitation phase (around Step 7 on-

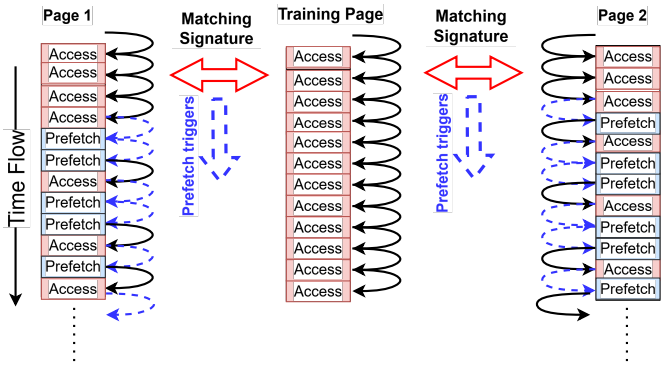

Figure 5: Bank Mirroring RowHammer attack using multiple pages

ward), where accesses are performed on two target pages in an alternating fashion. In particular, Step 7 and Step 9 alternate accesses between `arr1` and `arr2`, matching the learned stride behavior and causing the prefetcher to speculatively generate additional memory requests on both pages.

By continuously repeating this alternating pattern (Steps 7–12), the prefetcher amplifies the observed access stream by issuing additional microarchitectural memory requests beyond the program's explicit loads. While the program directly accesses only a small subset of memory locations, the hardware prefetcher effectively expands this footprint, increasing the rate of DRAM row activations. Since these prefetch-generated accesses occur entirely within microarchitectural structures, they remain invisible to software-level tracing mechanisms and incur no explicit instruction overhead, thereby enabling efficient multi-row activation that can facilitate Rowhammer effects.

4. Evaluation

4.1. Methodology

Using the microarchitectural simulator ChampSim, 2-billion-instruction SimPoints from each benchmark were executed, during which memory access patterns were recorded. In addition to access traces, standard performance and prefetching-related accuracy metrics were collected, with adjustments applied where necessary to account for IPC variations across configurations. To ensure the fidelity of the DRAM behavior modeled in ChampSim, results were cross-validated against the well-established Ramulator2 DRAM simulator [25].

Our simulated system (as shown in Table 1) models a 4 GHz out-of-order processor representative of modern client-class architectures. The memory subsystem consists of a single 8 GB DDR4 DIMM operating at 3200 MT/s using cache-line interleaving (CLI) mapping to enable bank-level parallelism. Unless otherwise specified, one processor core with its corresponding cache hierarchy is modeled.

Multiple samples were taken for each program using different randomized virtual-to-physical address mappings to capture variability introduced by memory allocation.

We use *hammer rate* as the accumulation of disturbances upon a single row in a DRAM bank by neighbors without a refresh, mitigative refresh, or access. Unnormalized, this is equivalent and comparable to the *hammer count* (HC) metric used in prior works.

Table 1: System Configuration

CPU Parameter	Value	DRAM Parameter	Value
Frequency	4 GHz	Frequency	3.2 GHz
L1I Cache Size	32 KB	Capacity	8 GB
L1D Cache Size	48 KB	Channels	1
L2 Cache Size	1 MB	Ranks	2
L3 Cache Size	2 MB	Banks	8
Cache Line Size	64 B	Rows	64 K
Page Size	4 KB	Columns	1 K

Table 2: Prefetcher Configurations

Prefetcher	L1D	L2	L3
VA-AMPM	None	VA-AMPM	None
Berti	Berti-L1D	Berti-L2C	Next Line
Bingo	Bingo	Next Line	Next Line
Bouquet	Bouquet-L1D	Bouquet-L2C	None
SPP	None	SPP	None

4.2. Prefetcher Configurations

We evaluate multiple state-of-the-art hardware prefetchers, including VA-AMPM [26], Berti [27], Bingo [28], Bouquet [29], and SPP. Each prefetcher is integrated into the cache hierarchy according to its original design. All other system parameters remain constant across experiments to enable fair comparison of prefetch-induced activation behavior. These configurations can seen in Table 2

4.3. Results

The hammering rate across different SPEC2017 benchmarks was evaluated both with and without prefetching enabled. Fig. 7 illustrates the impact of prefetching on the maximum

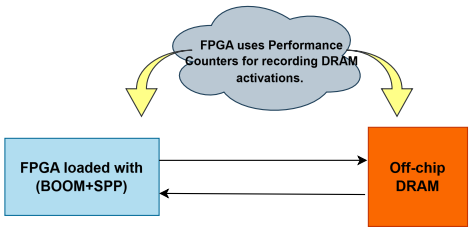

Figure 6: Real System Implementation on FPGA

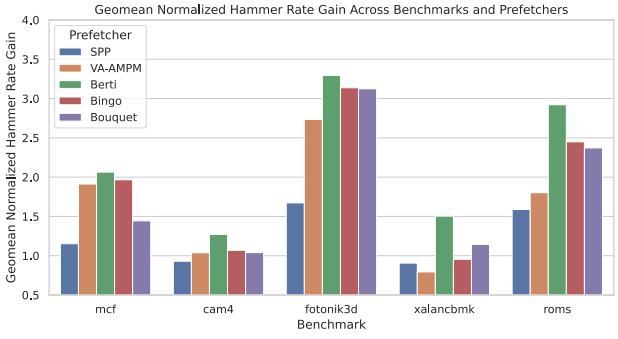

Figure 7: The multiplicative increase in hammer rate when using each prefetcher for each benchmark, normalized to the IPC speedup induced by the prefetcher.

observed hammer rates for each impacted benchmark, independent of IPC improvements. Specifically, we report the mean increase in the highest hammer rate relative to the non-prefetching baseline for each prefetcher configuration. The results indicate that prefetching mechanisms can significantly amplify hammering activity, irrespective of their effect on overall performance.

Furthermore, Fig. **??** shows the increase in hammering activity achieved by the bank-mirror attack in our simulation framework. The figure reports the number of hammers received by the target row in the mirror page across all DRAM banks. Banks 0, 1, and 2 are directly accessed by the attacker to train and trigger the prefetcher, while activations in banks 3–7 are generated exclusively by prefetcher-issued requests. These results demonstrate that hardware prefetchers can substantially amplify DRAM row activations beyond those explicitly generated by software, potentially increasing the severity of RowHammer-induced disturbances. The magnitude and distribution of activations vary across prefetchers due to differences in their prediction and confidence mechanisms. SPP's signature-based model with aggressive confidence reinforcement produces concentrated activation behavior, whereas Berti's reliance on temporal correlation yields a smoother distribution. Bingo tracks multiple overlapping patterns, spreading activations across more banks but with lower intensity per bank. In contrast, VA-AMPM and Bouquet do not generate additional hammering activity under our workload because their triggering conditions are not exercised. Importantly, the attack is not specific to SPP; it exploits a common characteristic of modern prefetchers that learn recurring access patterns and increase prediction confidence through repeated exposure. As a result, any prefetcher capable of learning repetitive access streams and applying that behavior across memory regions may be susceptible to similar activation amplification, with differences primarily affecting the extent rather than the feasibility of the attack.

4.4. Real System Implementation (Work in Progress)

As part of future work, we plan to develop and deploy a full-system FPGA-based platform to enable controlled generation of memory access patterns for training the SPP prefetcher and evaluating its ability to induce structured activation patterns on physical DRAM modules. We implemented SPP in Scala using the Chisel hardware construction framework and generated RTL for hardware-level experimentation. Due to the limited visibility of prefetchers from the software space, real-system evaluation will be performed via an FPGA platform equipped with real off-chip DRAM (as shown in Fig. 6), which also enables proper analysis and attribution of prefetcher-driven behavior. Furthermore, a comprehensive end-to-end evaluation requires modifications to both the prefetcher and memory controller to expose the internal events necessary for attack attribution and analysis, and the FPGA platform provides the flexibility required for such instrumentation. We integrate SPP within a RocketChip-based system featuring a BOOM core, allowing controlled evaluation of prefetcher training and its use as a mechanism for generating disturbance-oriented memory access patterns.

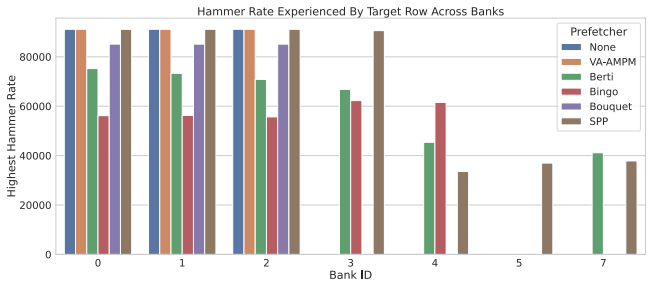

Figure 8: The hammer rate of the target row of our bank-mirroring hammer attack on our simulated system. The accesses beyond the third bank are generated entirely from the corresponding prefetcher's incorrect prefetches.

The BOOM core employs a two-level cache hierarchy, with SPP placed at the lower level of the memory hierarchy. Since FPGA implementations impose practical constraints on on-chip resource utilization, the cache hierarchy is provisioned with a moderate, representative LLC size rather than a large-scale last-level cache, ensuring a balanced trade-off between realism and implementability.

The last-level cache (LLC) is configured as an inclusive cache with a total capacity of 512,KB, organized as an 8-way set-associative structure with 64-byte cache lines. This setup will allow systematic characterization of the resulting DRAM behavior and its associated disturbance effects, with a particular focus on understanding whether prefetcher-driven access patterns can contribute to Rowhammer-style bit-flip phenomena. Using this full-system FPGA implementation as a real-system evaluation environment, we further aim to explore a broader class of prefetcher-driven attack mechanisms, systematically evaluating how different prefetching strategies can be leveraged to induce and amplify Rowhammer-style effects.

5. Conclusion

Prefetchers improve system performance by increasing IPC through speculative memory accesses. However, these additional accesses can also lead to a higher number of DRAM row activations, potentially exacerbating disturbance effects. In particular, a well-trained prefetcher may unintentionally amplify activation patterns in a manner that contributes to Rowhammer-like behavior. In this work, we investigate this interaction and aim to validate these effects through a real-system FPGA-based implementation, enabling more precise characterization of prefetcher-induced DRAM disturbances.

6. Acknowledgements

Portions of this research were conducted with the advanced computing resources provided by Texas A&M High Performance Research Computing.

The authors acknowledge the support from the Purdue Center for a Secure Microelectronics Ecosystem [CSME#210205].

References

[1] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 361–372.

[2] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *CoRR*, vol. abs/1904.09724, 2019. [Online]. Available: http://arxiv.org/abs/1904.09724

[3] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 638–651.

[4] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 245–261.

[5] P. Frigo, E. Vannacc, H. Hassan, V. v. der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "Trrespass: Exploiting the many sides of target row refresh," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 747–762.

[6] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "Zenhammer: Rowhammer attacks on amd zen-based platforms," in *Proceedings of the 33rd USENIX Security Symposium (USENIX Security '24)*. Philadelphia, PA, USA: USENIX Association, 2024, pp. 1615–1633. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/jattke

[7] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-Double: Hammering from the next row over," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3807–3824. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/kogler-half-double

[8] J. Kim, S. H. Pugsley, P. V. Gratz, A. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[9] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[10] W. Chen, S. Tang, Y. Tang, X. Luo, Y. Zhang, and W. Qiang, "ρHammer: Reviving RowHammer Attacks on New Architectures via Prefetching," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2025, pp. 1433–1447.

[11] M. Conti, S. Dragoni, V. Leschke, Y. Jang, and M. Franz, "Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2020, pp. e.g., 201–215.

[12] e. a. Zhang, "Ghostknight: Exploiting speculative loads for dram rowhammer effects," in *Proceedings of the 2020 ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2020.

[13] e. a. Tobah, "Spechammer: Leveraging transient execution to amplify rowhammer attacks," in *2022 IEEE Symposium on Security and Privacy (S&P)*, 2022.

[14] K. Loughlin, S. Saroiu, A. Wolman, Y. A. Manerkar, and B. Kasikci, "Moesi-prime: Preventing coherence-induced hammering in commodity workloads," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 670–684. [Online]. Available: https://doi.org/10.1145/3470496.3527427

[15] I. Kang, W. Wang, J. Kim, S. van Schaik, Y. Tobah, D. Genkin, A. Kwong, and Y. Yarom, "Sledgehammer: Amplifying rowhammer via bank-level parallelism," in *33rd USENIX Security Symposium (USENIX Security '24)*. Philadelphia, PA, USA: USENIX Association, Aug. 2024, pp. 1597–1614.

[16] A. Olgun, Y. C. Tugrul, N. Bostanci, I. E. Yuksel, H. Luo, S. Rhyner, A. G. Yaglikci, G. F. Oliveira, and O. Mutlu, "Abacus: All-bank activation counters for scalable and low overhead rowhammer mitigation," in *33rd USENIX Security Symposium (USENIX Security)*. Philadelphia, PA, USA: USENIX Association, 2024, pp. 1579–1596.

[17] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," in *Black Hat USA*, 2015, pp. 71–2.

[18] K. Bains, J. Halbert, C. Mozak, T. Schoenborn, and Z. Greenfield, "Row hammer refresh command," Patent, 2016.

[19] M. Merrell, L. Wang, S. Kalafatis, and P. V. Gratz, "Sppam: Signature pattern prediction and access-map prefetcher," *arXiv preprint arXiv:2602.04100*, 2026.

[20] J. Kim, E. Teran, P. V. Gratz, D. A. Jiménez, S. H. Pugsley, and C. Wilkerson, "Kill the program counter: Reconstructing program behavior in the processor cache hierarchy," *Acm Sigplan Notices*, vol. 52, no. 4, pp. 737–749, 2017.

[21] M. Shakerinava, M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Multi-lookahead offset prefetching," *The Third Data Prefetching Championship*, no. 2019, 2019.

[22] E. Bhatia, G. Chacon, E. Teran, P. V. Gratz, and D. A. Jiménez, "Enhancing signature path prefetching with perceptron prefetch filtering," *Delta*, vol. 1, p. 3, 2019.

[23] B. Grayson, J. Rupley, G. Z. Zuraski, E. Quinnell, D. A. Jiménez, T. Nakra, P. Kitchin, R. Hensley, E. Brekelbaum, V. Sinha, and A. Ghiya, "Evolution of the samsung exynos cpu microarchitecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 40–51.

[24] L. Xu, R. Yu, L. Wang, and W. Liu, "Memway: in-memorywaylaying acceleration for practical rowhammer attacks against binaries," *Tsinghua Science and Technology*, vol. 24, no. 5, pp. 535–545, 2019.

[25] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Ramulator 2.0: A modern, modular, and extensible dram simulator," 2023.

[26] Y. Ishii, M. Inaba, and K. Hiraki, "Access map pattern matching for data cache prefetch," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 499–500. [Online]. Available: https://doi.org/10.1145/1542275.1542349

[27] A. Navarro-Torres, B. Panda, J. Alastruey-Benedé, P. Ibáñez, V. Viñals-Yúfera, and A. Ros, "Berti: an accurate local-delta data prefetcher," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 975–991.

[28] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 399–411.

[29] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 118–131.