

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Charting the Cache Side-Channel Frontier: A Systematic Study of Eviction Set Construction on Apple Silicon

Han Wang, Yakun Wu, Yusi Feng, and Yinqian Zhang,
Southern University of Science and Technology

https://www.usenix.org/conference/usenixsecurity26/presentation/wang-han

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

Charting the Cache Side-Channel Frontier: A Systematic Study of Eviction Set Construction on Apple Silicon

Han Wang[†] Yakun Wu[†] Yusi Feng[†] Yinqian Zhang[†‡]
Southern University of Science and Technology

Abstract

Eviction sets constitute the fundamental primitive for conflict-based cache side-channel attacks, enabling unprivileged adversaries to induce contention deterministically. However, the security landscape of Apple Silicon remains largely underexplored due to the challenges of constructing these sets. We conduct a systematic reverse engineering study on the Apple M4 to uncover the root causes of these limitations, key microarchitectural impediments: an SLC victim buffer that obscures latency-based verification, a DRAM-to-Cache Prefetcher (DCP) that violates strict LRU replacement, and an opaque SLC allocation policy modulated by L2 Insertion/Promotion Vectors (IPV). Guided by these insights, we optimize construction strategies to overcome these impediments. Our enhanced Prime-Prune-Probe (PPP) achieves 88.2% success rate for L2 eviction sets. Furthermore, we employ a hierarchical filtering strategy to construct the first practical fine-grained SLC eviction sets on Apple Silicon. Our SLC optimal approach achieves a 90% success rate, unlocking fine-grained cross-cluster attack primitives previously deemed infeasible.

1 Introduction

The ARM architecture has evolved from a mobile-centric solution into a mainstream architecture spanning mobile, edge, and desktop domains [2, 22]. While ARM SoCs have long dominated the smartphone market, the transition of the Apple Mac platform to proprietary Apple Silicon has significantly accelerated ARM's proliferation in high-performance PCs [1, 3]. Given this widespread deployment, Apple M-series processors have emerged as high-value targets for side-channel attacks, prompting academia and industry to intensify scrutiny of their microarchitectural security [5, 14–16, 34, 39, 42].

In the realm of microarchitectural security, cache side-channels constitute a fundamental vector for modern processor attacks. These channels typically function in two ways: serving as covert exfiltration paths for transient execution attacks, in which speculatively encoded secrets are retrieved via latency differentials [6, 17]; or acting as independent side-channels that allow attackers to infer sensitive memory access patterns directly without reliance on speculation [16, 20, 43]. Central to the viability of these attacks is the construction of eviction sets, which serve as the requisite mechanism to induce contention and monitor the victim's cache activity.

While prior research on early Apple Silicon (e.g., M1/M2) [16,42] successfully constructed L2 eviction sets, primarily relying on Group Elimination (GE) [27, 35], the M4 introduces distinct microarchitectural advancements, specifically aggressive DRAM prefetching and complex cache replacement policies previously unobserved. These architectural shifts transform the M4 into a fundamentally more hostile environment. *Consequently, as empirically quantified in our baseline evaluation (Sec. 6.1), we identify two significant challenges hindering the construction of eviction sets on this platform:*

The *first* is the inefficacy of efficient algorithms on the L2 cache. In this strictly harder environment, state-of-the-art linear-time algorithms [27, 35] fail to construct valid eviction sets, rendering them ineffective and forcing a reliance on computationally prohibitive alternatives. The *second* is the infeasibility of constructing eviction sets for the System Level Cache (SLC). Even advanced methodologies explicitly tailored for non-inclusive hierarchies [44] fail completely at this cache level, rendering existing techniques inapplicable to the Apple SLC. These failures highlight a fundamental gap between generic algorithms and the specific, unknown microarchitectural behaviors of the M4.

Motivated by the unexpected divergence of standard eviction set algorithms on Apple Silicon, this study seeks to unravel the opaque microarchitectural interactions that defy established assumptions. We ask: *What obscure mechanisms within the M-series hierarchy actively obfuscate cache side-channel analysis?*

Through extensive microarchitectural characterization of the Apple M4, we expose a triad of hostile mechanisms that collectively destabilize eviction set construction: (1) an un-

‡Corresponding author.
†Affiliated with the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering.

characterized SLC victim buffer that obscures eviction signals by smoothing latency; (2) an aggressive DRAM-to-cache prefetcher (DCP) that induces speculative cache pollution during the construction process; and (3) an opaque SLC allocation policy strictly modulated by the upstream L2 state.

To decode these opaque behaviors, we employed a rigorous methodology. We determined the SLC Victim Buffer's precise timing thresholds by inducing massive and burst evictions. Despite the lack of physical address visibility, we utilized coarse-grained statistical profiling to confirm the DCP's existence and triggering patterns. Furthermore, demonstrating that the SLC is inextricably coupled with the L2, we devised the *Prime-Interfere-Probe* (PIP) and *Iterative Interfere-Probe* (I2P) protocols to decouple and reverse-engineer the complex L2 IPV dynamics and SLC replacement policies.

Guided by these insights, we systematically refined construction strategies in two stages. First, we optimized the L2 algorithms by disrupting the DCP's temporal and spatial locality and performing a comprehensive parameter sensitivity analysis. Second, we integrated these robust primitives with a hierarchical filtering strategy. This approach culminated in the first successful construction of fine-grained SLC eviction sets on Apple Silicon, achieving robust convergence where standard methods fail.

In summary, this paper makes the following contributions:

- **Reverse Engineering of Apple Silicon.** We conduct an in-depth reverse engineering of Apple M4, characterizing the SLC victim buffer, the DRAM-to-Cache Prefetcher (DCP), and the SLC allocation policy modulated by L2 Insertion/Promotion Vectors (IPV).
- **Tailoring L2 Construction to Apple Silicon.** Guided by our findings, we adapt existing algorithms to the microarchitectural characteristics of Apple Silicon. This Apple-specific adaptation mitigates convergence failures and achieves an L2 eviction-set construction success rate of 88.2%
- **First Practical SLC Eviction Set Construction.** Leveraging these insight, we successfully construct fine-grained eviction sets for the Apple SLC. We systematically evaluate various construction strategies, showing that our optimal approach achieves a success rate of 90%.

2 Background and Related Work

2.1 Cache Hierarchy of Apple M-Series Chips

Apple M-series silicon employs a heterogeneous architecture integrating Performance and Efficiency cores (e.g., 4 P-cores and 6 E-cores on the M4 chip). Each core features private Level 1 (L1) instruction and data caches, while cores within a single cluster share a unified Level 2 (L2) cache. The L2 cache maintains an inclusive policy with respect to the L1 caches in its cluster, ensuring that all data resident in L1 is also present in L2 [42]. Furthermore, the M-series integrates a System Level Cache (SLC), serving as a global coherence point for CPU clusters, the GPU, and other co-processors. The SLC implements a unique hybrid inclusion policy: it operates non-inclusively for CPU caches, but inclusively for GPU caches to facilitate data sharing [39]. Specifically, on the M4, the L2 cache is configured with 16 MiB and the SLC with 8 MiB, both utilizing a 128-byte cacheline size.

2.2 Eviction Set Construction Algorithms

We classify existing construction methods into generic algorithms and platform-specific optimizations.

2.2.1 Generic Construction Algorithms

Single-Holdout Method (SHM) [20] identifies redundant cachelines by temporarily withholding a single candidate and verifying if the conflict persists in the remainder. Redundant lines are discarded iteratively until the set converges to size W. With S sets and associativity W, the complexity is $O(S^2W^2)$.

Group Elimination (GE) [27, 35] enhances search efficiency by withholding groups of cachelines rather than individuals, leveraging the sparsity of conflicting lines within the candidate pool. If the conflict persists after removing a group, the entire group is permanently discarded. Follow-up work [32] reduces the algorithmic complexity from $O(SW^3)$ to $O(SW^2)$.

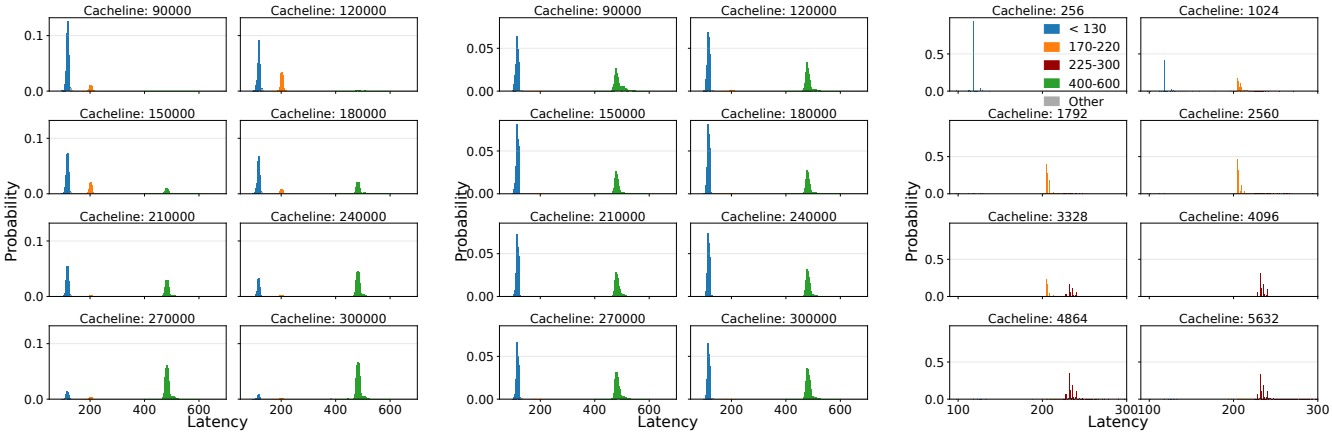
Conflict Testing (CT) [24, 27, 40] employs an incremental strategy by maintaining a cumulative testing set. Candidates that fail to trigger a conflict are appended to this testing set, whereas those successfully inducing a conflict are identified as valid members and isolated into the eviction set. This method applies to standard caches with $O(SW^2)$ complexity.

Prime-Prune-Probe (PPP) [25, 27] exploits approximate Least Recently Used (LRU) or Re-Reference Interval Prediction (RRIP) replacement policies using a reverse filtering mechanism. By priming with random addresses, pruning self-conflicts, and accessing the target, it identifies cachelines evicted by the target to form the eviction set. The theoretical complexity is $O(SW)$.

Prune+PlumTree (PPT) [13] optimizes *PPP* using a recursive binary strategy based on *Representatives*, where a single cacheline is selected to represent each of the S cache sets. It partitions the pool into lines evicted by specific *Representatives* and those not, efficiently identifying S eviction sets for all S representatives with $O(SW \log S)$ complexity.

L2CS [44] employs hierarchical filtering based on the subset relationship between L2 and LLC indices. In the absence of physical addresses or known slice hashes, it can utilize L2 eviction sets to screen candidates, efficiently pruning the search space. ZenLeak [37] also adopted this strategy.

Slice+Slice Baby [21] determines LLC slice mappings using a measurement gadget based on a comparator weird gate.



(a) Random access. (b) Flush after random access. (c) Random access with a page-size interval.

Figure 1: Fine-grained evolution of latency probability distributions across different test scenarios. Note that 90,000 and 300,000 cachelines correspond to approximately 11 MiB and 36.6 MiB, respectively.

Combined with the known slice hash, it recovers slice information for a page to filter the candidate set, enabling eviction set propagation under non-linear slicing functions.

Write+Write [33] leverages the write-after-write latency leakage in Intel chips to rapidly detect conflicts between two addresses. This enables a bottom-up construction approach where addresses are individually confirmed and added, significantly accelerating the process.

2.2.2 Optimizations for Apple Silicon

iLeakage [16] optimized GE for Apple M1. It accelerated reduction by utilizing an associativity upper bound (k) to discard multiple redundant bins per iteration. It also enhanced backtracking by refilling groups of candidates to counter noise.

S2C [42] addressed the uncertainty of the L2 inclusion policy on M1. To prevent the target from residing solely in L1, it constrained candidates to share the target's L1 set index. This enforces L1 eviction, ensuring the L2 eviction set functions correctly under both inclusive and exclusive policies.

3 Microarchitectural Characterization

This section characterizes the hierarchical cache latency and capacity on the Apple M4. Precise latency thresholds are prerequisites for building efficient eviction sets, since boundary errors inflate false positives and negatives.

Timing Measurement. We measure cycle counts via fenced reads of the PMU register `S3_2_c15_c0_0`, establishing a measurement overhead of ≈ 98 cycles [15, 28].

Experimental Configurations. To isolate specific microarchitectural behaviors, we devised three distinct benchmarks. In all configurations, we sweep the working set size (WSS) across a broad range to identify capacity thresholds:

- **Config A (Dense Random Access):** Random accesses with 128 B (cacheline) stride over multiple rounds to characterize baseline latency and effective capacity.
- **Config B (Flush-Based Random Access):** Extends Config A with an immediate cache flush after each access.
- **Config C (Sparse Conflict Access):** Random accesses with 16 KiB (page) stride to constrain set mappings, inducing high contention to expose eviction policies and victim buffers.

3.1 Fine-Grained Latency Analysis

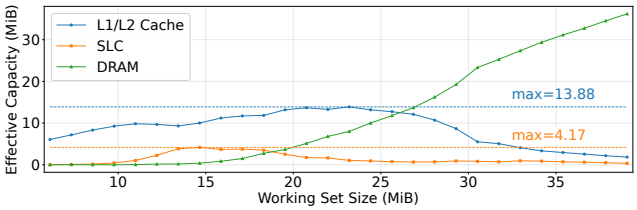
Using **Config A**, we tracked latency distributions derived from individual memory accesses to map distinct intervals to hardware events (Fig. 1a). We categorize the spectrum into three primary regions:

- **L1/L2 hits:** Core-private cache accesses (< 130 cycles).
- **SLC hits:** 170–300 cycles.
- **DRAM hits:** Main memory accesses (400–600 cycles).

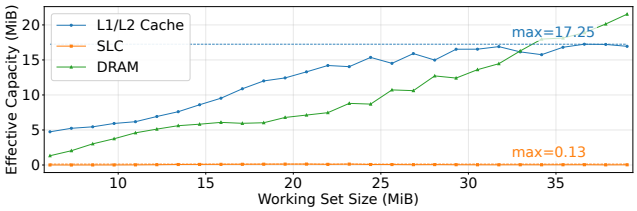
Based on this distributional analysis, we further evaluated the effective utilization of the SLC. Although the M4 chip has a nominal SLC capacity of 8 MiB, under these randomized patterns, we measured a maximum effective capacity of only 4.17 MiB. This corresponds to a utilization rate of approximately 52.13%, which is significantly lower than the 86.75% utilization observed for the L2 cache under identical conditions (as shown in Fig. 2a). **This implies the existence of specific internal mechanisms within the SLC that limit its effective capacity under random access patterns.**

Furthermore, a significant anomaly emerges when the WSS exceeds approximately 180,000 cachelines: while core-private cache hits persist, SLC hits virtually vanish. This suggests that in this region, the SLC is likely being effectively bypassed or is suffering from severe thrashing.

Observation. Random access yields low SLC utilization. Notably, exceeding specific size thresholds causes effective utilization to collapse to near zero.



(a) Random access.



(b) Flush after random access.

Figure 2: Effective capacity (WSS × Probability) for each memory hierarchy level versus total working set size.

3.2 Discovery of DRAM-to-Cache Prefetcher

To eliminate cache residency effects and establish a consistent DRAM access baseline, we employed **Config B**. Theoretically, the forced invalidation should ensure subsequent accesses miss the cache, resulting in a unimodal high-latency distribution characteristic of DRAM (400–600 cycles).

However, empirical results diverge significantly from these expectations. As illustrated in Fig. 1b, a distinct low-latency peak (< 130 cycles) persists despite the forced evictions. Furthermore, Fig. 2b shows that the effective utilization of the core-private cache exceeds 16 MiB. This low-latency profile strongly suggests the presence of an aggressive **DRAM-to-Cache Prefetcher (DCP)**. It appears this mechanism can identify patterns even under randomized execution, preemptively reloading data into the private cache hierarchy and effectively masking the inherent DRAM latency.

Observation. The anomalous residency of low-latency accesses implies that a prefetcher actively retrieves evicted data back into the cache hierarchy.

3.3 Discovery of the SLC Victim Buffer

To investigate data eviction from the L2 cache to the SLC, we transitioned to **Config C** (16 KiB stride). This modification is designed to induce set conflicts within the L2 cache, thereby maximizing the probability of evictions to the SLC.

Theoretical Analysis. Assuming a standard 16 MiB L2 cache with 16-way associativity and 128-byte lines, accessing memory with a 16 KiB interval restricts the lower-order index bits. This maps addresses to a sparse subset of sets (approximately 64). This extreme mapping imbalance rapidly exhausts the associativity of these sets, triggering frequent misses and forcing substantial data eviction to the SLC.

Experimental Results. The results reveal structural characteristics within the SLC hierarchy. As illustrated in Fig. 1c, as the working set size increases, the previously monolithic SLC latency interval bifurcates into two distinct regions:

- $Peak_1$ **(170–220 cycles):** Standard SLC hit latency.
- $Peak_2$ **(225–300 cycles):** Accesses to data evicted from the main SLC. These lines exhibit elevated latency, indicating they are served by a secondary buffering structure.

Observation. The SLC latency interval can be divided into two distinct segments by a threshold of 225 cycles.

This evolution provides compelling evidence for the existence of an **SLC Victim Buffer (VB)**. When data is evicted from the main SLC array, it is not written back to DRAM immediately but is staged in this buffer. The latency observed in $Peak_2$ characterizes accesses to this structure, effectively smoothing performance jitter during high-eviction scenarios.

Takeaway. The SLC Victim Buffer masks eviction events. To ensure algorithmic convergence, we must enforce a latency threshold to strictly exclude these buffered accesses from the candidate pool.

4 Unveiling DRAM-to-Cache Prefetcher

The latency benchmarks in Sec. 3.2 revealed an anomaly where flush-based experiments exhibited latencies significantly lower than DRAM access times. In this section, we identify the root cause as the **DRAM-to-Cache Prefetcher (DCP)**, a mechanism that predicts future memory accesses and speculatively loads data from DRAM into core-private caches. We characterize its triggering patterns and spatial constraints, as this prefetcher disrupts replacement policies and poses a critical challenge to eviction set construction.

4.1 Behavioral Characterization of DCP

To verify the triggering conditions of the DCP, we designed a more rigorous set of controlled experiments. We selected 32 memory pages and targeted the first cacheline of each page for testing. To eliminate interference from other prefetchers, we devised two non-sequential access schemes:

- **Linear Shift Mode** (Pseudo-random): Access indices are generated using a linear congruential generator algorithm:

$Index = (i \times 13 + 167) \mod 32$. This pattern is deterministic but disrupts continuous address strides.

- **Fully Random Mode**: Access indices are generated via a random function to eliminate address predictability.

In each trial, a flush instruction was executed immediately post-access to evict the line to DRAM. Additionally, a system-wide flush was performed between rounds to ensure a cold cache state. The experiment comprised 100 iterations, each consisting of 300 access rounds to the address pool, recording the hit status of each page.

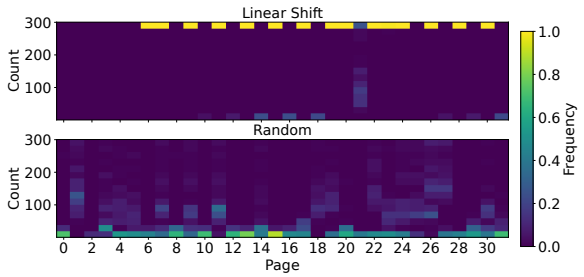


Figure 3: Heatmap characterizing DCP behavior. The X-axis represents the target Page ID (0-31). The Y-axis (**Count**) represents the total number of cache hits observed for that page within a single 300-access trial (aggregated into bins of 20). The color intensity (**Frequency**) indicates how often this specific hit count occurred across 100 independent trials. (Top) Linear Shift mode. (Bottom) Fully Random mode.

The results reveal distinct anomalies in Fig. 3. Despite the forced flush mechanism and non-sequential patterns, pronounced core-private cache hit signals persist in the heatmap. In Linear Shift Mode, dense yellow bands appear at the top for specific target pages. This indicates that the DCP successfully modeled the linear shift pattern for these pages, actively prefetching the target pages. Even in Fully Random Mode, a substantial number of cache hits persist.

These findings confirm the existence of the DCP. They further indicate that intensive DRAM access streams induce the DCP to load data into L1/L2 caches, even under fully random access sequences. Subsequent analysis in Sec. 6 will demonstrate how this behavior impacts eviction set construction.

4.2 The Minimal Triggering Set

To characterize the DCP triggering logic and isolate the minimal access pattern required for activation, we devised a reduction-based reverse engineering protocol. This protocol filters a large candidate address pool to identify the critical set of cachelines necessary to trigger prefetching for a specific target. The procedure unfolds in four steps:

Step 1. Target Selection: The first cacheline of a specific page is designated as the monitoring target.

Step 2. Iterative Reduction: We initialize the address pool with the first cachelines of all pages and apply a reduction strategy similar to eviction set construction:

- A non-target cacheline is temporarily removed (skipped) from the current access sequence.
- We access the remaining pool and verify if the target is prefetched (indicated by a low-latency hit).
- If prefetching fails, the removed line is deemed *critical* to the trigger condition and is restored to the pool. Conversely, if prefetching persists, the line is considered *redundant* and is permanently discarded.

Step 3. Statistical Validation: Step 2 is repeated 100 times for each target to mitigate noise.

Step 4. Pattern Extraction: We calculate the count with which each cacheline appears in the final minimal set (visualized in Fig. 4). Lines exceeding an empirical threshold of 30 occurrences are identified as critical dependencies.

The analysis of critical triggering pages, presented in Fig. 4, reveals two pivotal characteristics of the DCP's activation:

Non-uniformity of Prefetching. Not all targets trigger prefetching reliably; the omission of subplots for non-triggering targets indicates that for certain pages, no stable prefetching was observed under the tested constraints.

Sparse Triggering Pattern. The DCP does not require high-volume continuous streams. For susceptible targets (e.g., Targets 6, 16, 18, 22, 31), a minimal sequence of just three specific precursor cachelines is sufficient to activate the mechanism and prefetch the subsequent target.

> **Observation.** The DCP is triggered by a minimal sequence of just 3 precursor accesses.

4.3 Spatial Triggering Patterns

Building on the finding that a minimal sequence of four cachelines (including targets) activates the DCP, we employed an exhaustive enumeration strategy to characterize the precise geometric relationships within these triggering quadruplets. We systematically varied the spatial granularity from 128 B to 16 KiB and tested all possible quadruplet combinations to determine whether accessing a precursor triplet would trigger the prefetching of a fourth line into the core-private cache.

For all successful triggering sequences, we calculated the adjacent differences between sorted cacheline indices. As illustrated in Fig. 5, the results reveal a distinct pattern: the critical address deltas are not randomly distributed but are strictly constrained to values of 1, 2, 4, and 8. This consistency holds across all tested granularities.

We hypothesize that this 2^n interval pattern is non-arbitrary and reflects underlying hardware constraints governing DRAM geometry and memory controller behavior. Specifically, these intervals likely ensure that the discontinuous cachelines map to the same DRAM Row, thereby residing

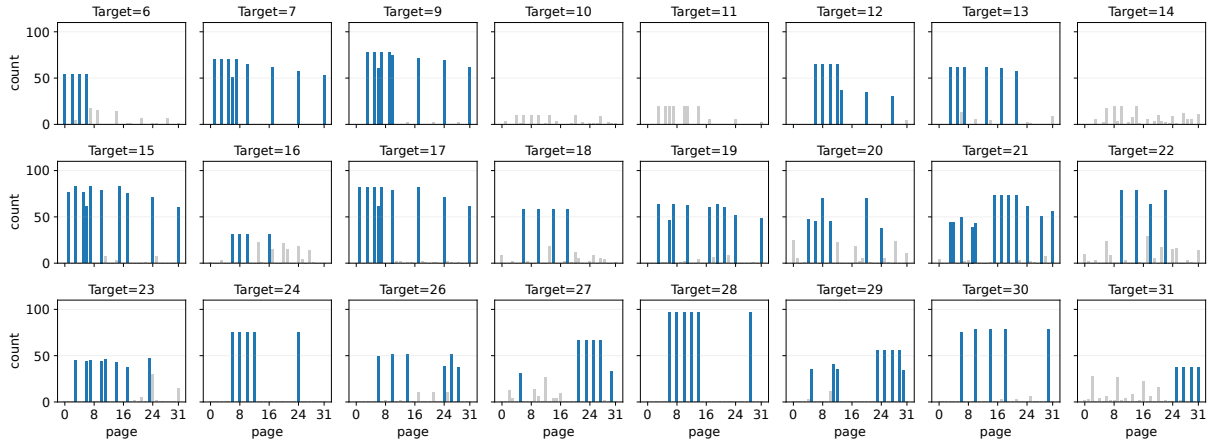


Figure 4: Identification counts of DCP-triggering pages across 32 targets during the reduction phase. The Y-axis (**Count**) represents the number of times a specific page was identified as a critical dependency. Blue bars denote critical pages exceeding the threshold; grey bars represent noise. (Non-triggering targets are omitted).

within a single Row Buffer.

The DCP appears designed to exploit this Intra-row Spatial Locality. Upon detecting sparse accesses within a single row that conform to these specific spacings, the mechanism identifies the row as high-utility. Consequently, it aggressively prefetches subsequent data from that row (e.g., the target line) into the core. This strategy aims to maximize Row Buffer hit rates, thereby amortizing the high latency associated with row activations and cross-row accesses.

Observation. Triggering sequences are strictly constrained to 2^n spatial strides.

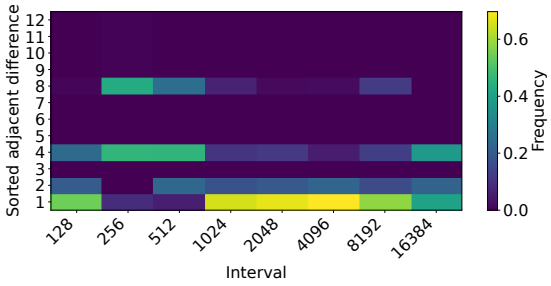


Figure 5: Distribution of DCP-triggering address deltas across varying base intervals.

Distinction from known prefetchers. The DCP behavior we observe on the Apple M4 does not fit the known classes of commercial hardware prefetchers. It is not a conventional stride [8] or stream [29] prefetcher: those mechanisms infer regular address progressions from recent accesses, whereas the DCP is triggered by power-of-two-spaced accesses and may reload previously evicted data from DRAM into the cache hierarchy. It also differs from content-based prefetchers, such as data-memory-dependent or pointer-based prefetchers [5,34,36], because its trigger is not the loaded data value. Nor is it equivalent to side-channel mechanisms such as PrefetchX [7], which concurrently issue requests to both DRAM and the LLC without any stride-based constraints. Finally, unlike DRAMA-style effects, which rely on passive DRAM row-buffer locality [23], the DCP actively changes cache residency by bringing evicted lines back into the cache. Existing systematic characterization frameworks such as FetchBench [30] did not identify an analogous mechanism, and repeating our DCP-triggering experiment on Intel-285H/10700 did not produce comparable low-latency peaks. We therefore treat the DCP as a distinct microarchitectural mechanism whose behavior must be explicitly accounted for during eviction-set construction.

4.4 Mitigation of DCP Interference

To ensure the construction of noise-free, pristine eviction sets, we investigated strategies to inhibit DCP activation by disrupting the temporal locality and spatial alignment inherent to its triggering conditions. We devised and evaluated two specific interference mechanisms across varying spatial granularities and sequence lengths:

- **Temporal Perturbation.** We injected artificial delays into the access sequence by executing empty loops (`NOPs`) between consecutive memory accesses. We evaluated two intensity levels: `NOPs=100` and `NOPs=1000`, representing the loop count.
- **Spatial Offset.** To break the strict spatial alignment constraints identified in Sec. 4.3, we introduced unaligned random offsets to the addresses of the access sequence (`Offset Enable = 1`).

The effectiveness of these strategies is illustrated in Fig. 6.

The results reveal a distinct dichotomy in DCP behavior depending on the address interval:

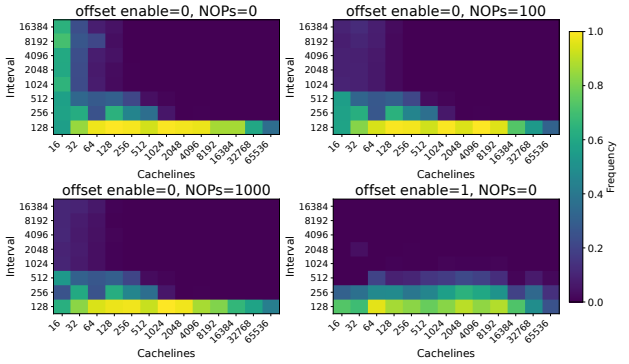


Figure 6: Impact of temporal delays and address offsets on DCP triggering frequency.

Ineffectiveness at Small Intervals. For compact access patterns, the prefetching proves highly resilient. As indicated by the persistent bright yellow band in the lower regions of the heatmap, neither spatial offsets nor temporal delays (up to 1000 `NOP`s) could effectively suppress core-private cache hits. We hypothesize that this persistent activity is attributable to a standard next-line prefetcher, operating independently of the DCP and remaining unaffected by the targeting perturbations.

Successful Suppression at Large Intervals. Conversely, in the sparse regions (Interval > 512), the DCP exhibits significant sensitivity to temporal perturbation. As the delay is inserted, the prefetching frequency drops precipitously.

Feasibility for Eviction Set Construction. The prefetching observed at dense granularities does not impede our objective, as construction employs 16 KiB page-aligned intervals to restrict mappings to a sparse subset of sets. At this granularity, temporal perturbation effectively suppresses DCP. However, a critical caveat remains: as the candidate pool expands, minimal residual noise aggregates to non-negligible magnitudes. We will discuss this in Sec. 6.

Takeaway. Employing large address intervals and limiting the candidate pool greatly mitigates DCP interference.

4.5 Implications on Flush+Reload Decoders

We provide empirical validation of our findings by analyzing the microarchitectural implications of the M4 DCP on covert channel decoding. We emphasize that our focus is on demonstrating how the DCP inherently disrupts the decoders used in the evaluated artifact [4]. Specifically, we examine a Flush+Reload decoder artifact that employs a Linear Congruential Generator (LCG) style permutation, defined by $Index = (i \times 167 + 13) \pmod{256}$, designed to randomize the access sequence and avoid conventional stride prefetchers.

While this pseudo-random pattern successfully evades traditional stride-based prefetching, we observed that on the Apple M4, this specific reload sequence itself triggers the DCP. As a result, lines intended to remain flushed are aggressively refetched into the cache hierarchy during the decoding phase. This inadvertent prefetching masks the critical side-channel signals, effectively corrupting the covert channel and causing the decoding process to fail.

5 Unveiling L2 Dynamics and SLC Allocation

Beyond prefetching anomalies, our measurements indicated that the effective SLC capacity under random access patterns is substantially lower than its nominal value. This suggests a restrictive allocation policy that filters incoming data. Since the SLC functions as a victim cache for the L2, characterizing this behavior requires a holistic analysis of the L2-SLC hierarchy. In this section, we investigate the intricate coupling between the **L2 Insertion/Promotion Vector (IPV)** [12] (responsible for L2 insertion priorities) and the opaque **SLC Allocation Policy** [10], determining how specific access contexts govern the admission of L2 victims into the SLC.

5.1 Prime-Interfere-Probe (PIP)

To characterize these mechanisms, we propose the Prime-Interfere-Probe (PIP) protocol (Fig. 7). Designed to decouple access patterns, data structures, and prefetchers, PIP utilizes two 16 MiB regions ($Region_1$,$Region_2$), matching L2 capacity, to model victim-eviction interactions across three stages:

Stage 1. Prime: We access $Region_1$ to populate the L2 with a known state. To evaluate IPV assignment, we employ two strategies: *Prime(S)* (linear strided access) and *Prime(R)* (randomized access with read-after-write dependencies).

Stage 2. Interfere: We access $Region_2$ to evict $Region_1$, employing three modes to isolate prefetcher noise: *Interfere(S, offset)* (strided array access), *Interfere(R, offset)* (random array access), and *Interfere(R, PC)* (pointer chasing). The latter is specifically introduced to examine the Data Memory-Dependent Prefetcher (DMP) [5, 34].

Stage 3. Probe: We randomly re-access $Region_1$ (*Probe(R)*) to measure the latency of each cacheline.

We designed eight controlled experiments (Exp 1–8), repeated iteratively to mitigate system noise:

- **Baseline (Exp 1):** Establishes the reference latency profile.
- **Self-Conflict Control (Exp 2–3):** Executes only the Prime and Probe stages to quantify internal self-eviction rates within $Region_1$, serving as a control group.
- **Cross-Conflict Evaluation (Exp 4–8):** Incorporates the Interfere stage. By permuting different Prime and Interfere patterns, we test the IPV priority of cachelines established under different patterns against varying types of interference.

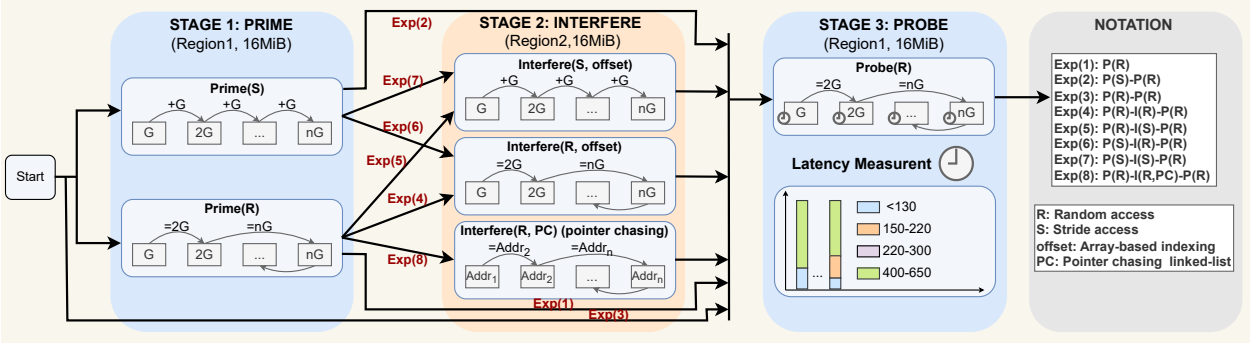


Figure 7: Schematic diagram of the three-phase reverse engineering protocol: Prime-Interfere-Probe (PIP).

Based on the PIP protocol, we analyzed the latency distributions obtained during the Probe stage, as illustrated in Fig. 8. By comparing the L2 hit rates (<130 cycles) across these experiments, we effectively reverse-engineer the rules governing the L2 IPV assignment.

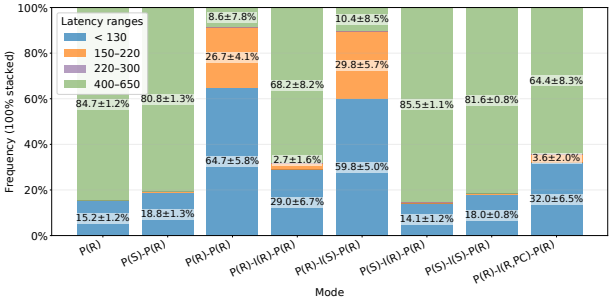


Figure 8: Latency distribution during the Probe phase across the 8 control experimental groups in the PIP protocol.

Impact of Access Patterns on IPV (Exp 2 vs. Exp 3). Comparing Exp 2 ($P(S) \rightarrow P(R)$) and Exp 3 ($P(R) \rightarrow P(R)$) reveals a significant disparity driven solely by the Prime pattern. Exp 3 (Random Prime) maintains a high hit rate of 64.7%, whereas Exp 2 (Stride Prime) yields only 18.8%. This confirms that the L2 controller assigns distinct IPVs based on access patterns:

- **Low IPV:** Strided accesses are flagged as eviction-prone, likely to mitigate cache pollution from streaming scans.
- **High IPV:** Random accesses with dependencies are treated as critical, receiving higher priority and residency weights.

Observation. IPVs strictly modulate residency: High IPV lines resist eviction, while Low IPV lines are susceptible.

IPV-Driven Replacement Policy (Exp 4 vs. Exp 5). In Exp 4 ($P(R) \rightarrow I(R) \rightarrow P(R)$), random interference reduced the $Region_1$ hit rate from 64.7% to 29.0%. Conversely, Exp 5 ($P(R) \rightarrow I(S) \rightarrow P(R)$), utilizing stride interference, maintained a high hit rate of 59.8%. This confirms a replacement policy bias: incoming stride streams are assigned Low IPV, failing to displace resident High-IPV data. Consequently, the controller preserves $Region_1$ content while discarding the incoming stride sequence from $Region_2$. Exp 6 and Exp 7 provide complementary evidence. When the prime phase uses stride accesses, the primed lines themselves are assigned Low IPV. Unlike the randomly primed lines in Exp 4 and Exp 5, these Low-IPV residents receive little replacement protection and are therefore more likely to be evicted by subsequent interference. Thus, the results of Exp 6 and Exp 7 are consistent with the same IPV-driven mechanism: stride streams both fail to evict High-IPV residents and produce Low-IPV residents that are easily discarded.

Takeaway. Random access is mandatory to force the L2 to assign High IPV. This priority ensures candidates achieve sufficient residency to displace the target.

Data Structure Efficacy (Exp 4 vs. Exp 8). Exp 4 (Base + Offset) shows a slightly lower mean L2 hit rate than Exp 8 (Pointer Chasing), at 29.0% versus 32.0%. Since their run-to-run variance overlaps, we do not claim a decisive advantage for either data structure. Instead, this result indicates that Base + Offset provides comparable eviction efficacy while avoiding the additional DMP-induced noise introduced by pointer chasing. In Exp 8, pointer chasing may trigger speculative DMP fetches assigned Low IPV, which are less effective at displacing resident High-IPV victims and can dilute eviction pressure. We therefore prefer Base + Offset for L2 construction because it is simpler, more controlled, and no worse in our measurements.

5.2 Iterative Interfere-Probe (I2P)

To characterize L2 post-eviction flow and reverse-engineer SLC allocation policies, we developed the *Iterative Interfere-Probe (I2P) Protocol* (Fig. 9). Building on Exp 5 ($P(R) \rightarrow I(S)$), I2P exploits a key architectural trait: cache maintenance instructions (e.g., `dc civac`) target core-private caches but do not invalidate copies resident in the System Level Cache

(SLC). The protocol executes in two phases:

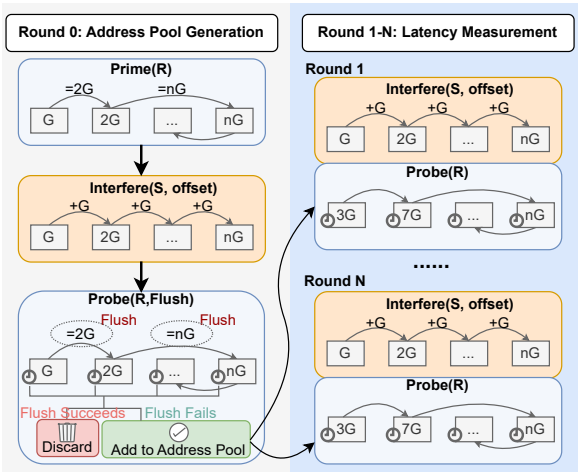


Figure 9: Schematic of Iterative Interfere-Probe (I2P).

Phase 1: SLC Resident Pool Isolation (Round 0). We initiate with $P(R)$ followed by $I(S)$. While strided interference generally fails to evict High-IPV L2 lines, capacity contention forces spills into the SLC. In the subsequent *Probe(R, Flush)* step, we execute a flush instruction on each address immediately prior to access, acting as a filter:

- **L1/L2 Resident:** The flush invalidates core-private lines, forcing a DRAM fetch (high latency). These are discarded.
- **SLC Resident:** Lines evicted to the SLC survive the core-private flush. Accesses yield medium latency (SLC hit). We aggregate these lines into the *SLC-Resident Pool*.

Phase 2: Retention Characterization (Rounds 1–N). We subject the SLC-Resident Pool to repeated $I(S) \rightarrow P(R)$ rounds, recording access latencies to characterize SLC data retention behavior over time.

To investigate the temporal evolution of L2 replacement policy and its SLC impact, we extended probing to 30 rounds across varying working set sizes (100k–300k lines). As shown in Fig. 10a, the L2 hit rate starts low (Rounds 1–2) but stabilizes into sustained residency. This implies the L2 controller employs a learning mechanism: repeated access/eviction cycles train the algorithm, dynamically elevating the IPV of these lines to secure long-term retention.

Observation. Repeated evictions trigger a dynamic elevation of the cacheline IPV that prioritizes its retention.

Eventually, capacity pressure evicts even High-IPV lines. Cross-referencing SLC (Fig. 10c) and DRAM (Fig. 10b) heatmaps reveals a counter-intuitive trend: High-IPV evictions do not increase SLC residency but instead drive a sharp rise in DRAM hits. We classify the SLC allocation logic into two distinct strategies modulated by the L2 state:

- **Standard Allocation (Low L2 IPV):** Data is preserved in the SLC, following standard victim caching procedures.
- **De-allocation (High L2 IPV):** Data is inferred to have completed its utility lifecycle. Consequently, it is de-prioritized and rapidly propagated to DRAM.

Observation. SLC allocation is modulated by L2 IPV: High IPV lines trigger de-allocation, resulting in rapid eviction to DRAM.

To characterize hierarchy dynamics, we tracked the migration trajectory of each individual cacheline within the SLC-Resident Pool. By aggregating these per-line transitions, we constructed a State Transition Matrix (Fig. 11). This matrix maps the probability of data moving from a previous state (y-axis) to a current state (x-axis), revealing three insights:

Absence of SLC No-Allocation Strategy. The first row (Previous=L2) shows L2 evictions migrate almost exclusively to the SLC, not DRAM. This refutes the existence of a strict no-allocation (bypass) policy. The rapid DRAM propagation of High-IPV data stems from insertion at the LRU position, rather than a lack of allocation.

Takeaway. The absence of a strict no-allocation policy ensures L2 evictions reliably populate the SLC, enabling valid eviction set construction.

Invisibility of the Victim Buffer. The VB acts as a temporary staging area, becoming observable only when intense conflicts saturate the DRAM bandwidth (Sec. 3.3). Otherwise, evicted data drains immediately to DRAM.

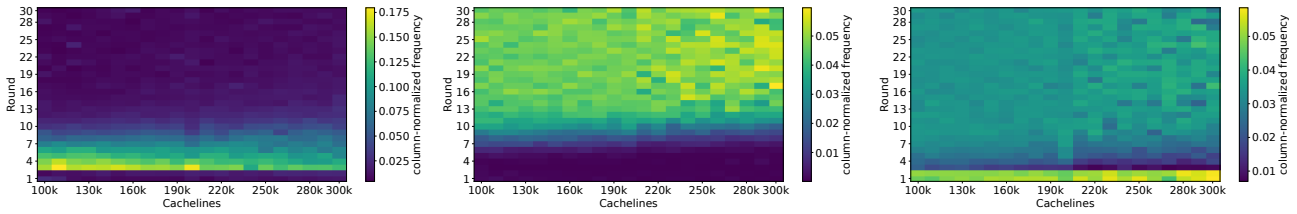
SLC-DRAM Thrashing. The matrix highlights frequent bidirectional transitions between the SLC and DRAM, indicative of severe cache thrashing.

Table 1: Top-5 SLC/DRAM sequences during rounds 16–20 (0 = SLC hit, 1 = DRAM access).

ID	Pattern	Count	Fraction
21	010101	58145	0.143285
42	101010	57363	0.141357
0	000000	55921	0.137804
20	010100	22032	0.054293
10	001010	21095	0.051984

To decode the SLC replacement policy, we analyzed residency patterns (Rounds 16–20) as binary sequences (0: SLC, 1: DRAM; see Table 1), yielding two findings:

- **Non-Random Replacement:** The access sequences exhibit high regularity, effectively refuting the hypothesis of a purely random replacement policy in previous work [39].
- **Grace Period:** The regularity suggests a grace period, ensuring that newly allocated data persists for at least one interference cycle, mitigating thrashing under bursty traffic.



(a) L2 (b) DRAM (c) SLC

Figure 10: Evolution of L2, SLC, and DRAM hit rates collected by the I2P protocol across varying rounds.

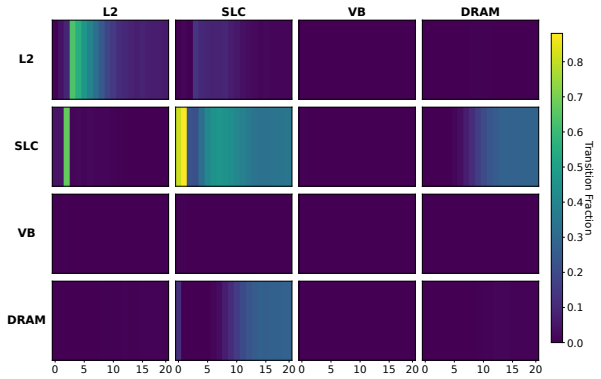


Figure 11: Cache hierarchy state transition probability matrix. Axes denote data locations in the previous (vertical) and current (horizontal) rounds.

6 Optimization and Evaluation

We systematically evaluate methods for constructing eviction sets. We begin by quantifying the limitations of standard algorithms to establish a baseline and motivate the need for optimization. Subsequently, we refine L2 construction by isolating four critical variables: **access randomization**, **access frequency**, **robustness mechanisms**, and **candidate pool size**. In each sensitivity analysis, all other variables are maintained at their empirically determined optimal states. These optimizations are subsequently integrated into the hierarchical construction of SLC eviction sets. Finally, we characterize the latency properties and stability of the generated sets.

Evaluation Metrics. We define a successful eviction set based on its ability to force the target line out of the cache. Specifically, success requires the target to exhibit next-level latency in at least **20 out of 30 trials**. We define *strict success* (default) as finding a set size equal to the associativity (W), whereas *loose success* permits sizes $< 2W$. Elapsed time measures the duration of a single complete execution.

Experiment Setup. All evaluations were performed on an Apple M4 Mac mini. We utilize the hardware PMU (register `S3_2_c15_c0_0`) for high-precision timing, which is consistent with prior research [14, 15, 28]. For environments with no PMU access, a helper thread incrementing a shared atomic counter can alternatively serve as a timing proxy [11].

6.1 Limitations of Generic Approaches

To contextualize the challenges of cache side-channel attacks on Apple Silicon, we first evaluated the efficacy of standard, unoptimized eviction set construction algorithms.

Ineffectiveness of Efficient L2 Algorithms. A critical trade-off exists between algorithmic complexity and robustness. While the state-of-the-art *PPP* offers superior theoretical complexity ($O(SW)$) [25, 27] compared to *GE* ($O(SW^2)$) [27, 35], prior research on M-chips [16, 42] has predominantly relied on the computationally heavier *GE*.

Our baseline evaluation reveals the reason for this preference. We implemented a standard *PPP* algorithm and conducted 1000 trials. The algorithm achieved a negligible **1.60% success rate**, typically failing to converge to minimal sets due to noise intolerance. Even augmenting *PPP* with a *GE* post-pruning step only improved success to **21.90%**. These results confirm that specific L2 mechanisms reverse-engineered in previous sections actively disrupt the strict LRU-based assumptions required for *PPP* to function.

Gap in Fine-Grained SLC Eviction. The SLC acts as the shared medium for attacks traversing heterogeneous boundaries (e.g., CPU/GPU or P/E clusters). However, existing literature [39] is limited to monitoring coarse-grained SLC occupancy. This limitation creates a critical blind spot for security assessments. We attempted to bridge this gap by adapting the standard L2CS [44] methodology to the M4 SLC. This attempt proved futile: the algorithm yielded a non-convergent eviction set with **a 0% success rate**.

6.2 L2 Eviction Set Construction

As illustrated in Fig. 12, the evaluated algorithms utilize distinct filtering paradigms to isolate conflicting cachelines. The goal of L2 eviction-set construction is to identify a set of congruent cachelines that can evict the target cacheline from the L2 cache into the SLC. The **Single-Holdout Method (SHM)** and its optimized variant, **Group Elimination (GE),** employ reduction-based strategies, iteratively discarding redundant candidates (either individually or in groups) to distill a mini-

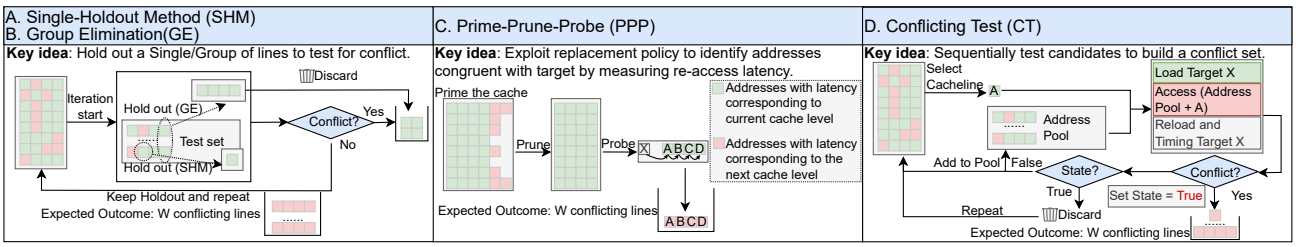


Figure 12: Schematic overview of four mainstream L2 eviction set construction algorithms.

Table 2: L2 construction performance vs. pool size. Parentheses (SHM) indicate loose success rate.

	CT		GE		PPP		SHM	
Size(MiB)	Success rate	Elapsed time(ms)	Success rate	Elapsed time(ms)	Success rate	Elapsed time(ms)	Success rate	Elapsed time(ms)
16	0.000	154.220	0.520	17.013	0.468	12.754	0.210(0.450)	1173.652
32	0.372	582.090	0.996	27.532	0.805	15.790	0.170(0.770)	20310.087
64	0.663	780.904	0.998	41.557	0.882	21.471	0.150(0.790)	113794.358
128	0.667	917.485	0.994	75.986	0.415	32.628	–	–
256	0.702	1137.482	0.998	146.700	0.081	44.049	–	–
512	0.698	1580.763	0.992	388.745	0.088	75.855	–	–
1024	0.699	2625.470	0.996	1022.379	0.099	120.535	–	–

mal eviction set. Conversely, **Conflict Testing (CT)** adopts an incremental approach, sequentially accumulating candidates until an eviction set is found. Finally, **Prime-Prune-Probe (PPP)** exploits cache replacement to identify specific lines displaced by the target access. We further refine *PPP*'s output using a lightweight *GE* step to guarantee convergence, which adds negligible latency due to the small candidate count.

6.2.1 Impact of Randomization

Guided by previous insights, we evaluated the impact of access randomization. As shown in Table 3, algorithms sensitive to replacement state (*CT* and *PPP*) exhibit severe degradation under sequential access, while *GE* remains unaffected.

This disparity stems from prefetcher interference. Sequential patterns activate the prefetcher, which disrupts the strict LRU replacement required by *PPP*. Furthermore, prefetching degrades the IPV of cachelines, preventing the eviction of the target in both *CT* and *PPP*. Randomization suppresses the prefetcher, restoring expected replacement behaviors. In contrast, *GE* disrupts spatial locality by rapidly discarding large chunks of candidates. This fragmentation effectively breaks the sequential patterns required for prefetcher activation.

6.2.2 Sensitivity to Access Frequency

We evaluated the impact of access frequency per candidate line (*# of maccess*) on success rates across 1000 trials (Fig. 13). *GE* exhibits superior robustness, achieving near-100% success with minimal accesses. Conversely, *PPP* displays high sensitivity; it requires sufficient access frequency to promote cachelines to high-priority states, preventing premature eviction during pruning. Precise calibration of access counts is prerequisite for effective *PPP* deployment.

Table 3: Impact of access pattern randomization on eviction set construction.

Algorithm	Random	Success rate	Elapsed time(ms)
CT	✗	0.056	338.382
CT	✓	0.942	738.849
GE	✗	0.990	42.052
GE	✓	0.998	40.304
PPP	✗	0.488	29.790
PPP	✓	0.842	19.675

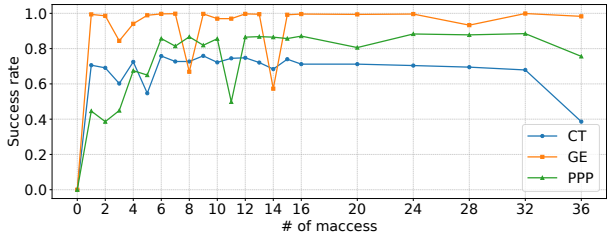


Figure 13: Relationship between L2 eviction set construction success rate and number of accesses.

6.2.3 Robustness Optimization Mechanisms

Single-pass evaluations are susceptible to false positives and negatives caused by system noise, hardware prefetching, and non-ideal replacement behaviors. To mitigate this, we inte-

grated algorithm-specific robustness strategies into each algorithm (performance comparisons in Table 4).

Table 4: Performance comparison of algorithms before and after introducing Backtracking mechanisms.

Algorithm	Robustness	Success rate	Elapsed time (ms)
CT	✗	0.530	294.893
CT	✓	0.704	730.980
GE	✗	0.899	39.186
GE	✓	0.997	41.658
PPP	✗	0.820	17.223
PPP	✓	0.874	20.621

GE: If a round fails to identify a removable group, the algorithm reverts to the previous state and reshuffles the traversal order rather than terminating [35]. This mechanism proves highly effective, boosting the success rate from 89.9% to 99.7% with a negligible timing penalty (≈2.5 ms).

CT: Over-collection occurs when the pool accumulates a full eviction set (W conflicting lines), rendering subsequent validation checks meaningless. Backtracking resolves this by rotating addresses from the pool back to the candidate tail until the target is no longer evicted. This mechanism improves success rates but significantly increases latency.

PPP: To counteract prefetcher-induced probabilistic noise, we implemented a Majority Voting strategy. A cacheline is pruned only if marked as evicted in more than half of the consecutive *Prune* rounds. This statistical filtering improves the success rate from 82.0% to 87.4% while maintaining a low overhead (≈20 ms) due to *PPP*'s high execution efficiency.

6.2.4 Impact of Candidate Pool Size

We evaluated the critical trade-off between address pool capacity and construction efficiency in Table 2. Due to SHM's prohibitive latency (>113s at 64 MiB) , we restricted its evaluation to 16–64 MiB with reduced sampling (100 trials vs. 1000 for others).

CT shows a positive correlation between pool size and success rate, stabilizing at ≈70%. The low success rates at 16–32 MiB stem from the algorithm's specific dependency on conflict density: it requires a baseline of $W-1$ cachelines to form a testing set for verifying new candidates. Consequently, the total candidate pool must harbor a minimum of $2W-1$ conflicting cachelines to facilitate the effective construction of a full eviction set.

Conversely, *PPP* excels with small pools (<64 MiB) but suffers a precipitous performance drop (88.2% → 8.8%) as the pool expands beyond 128 MiB. This anomaly is attributed to the interaction between large working sets and the DCP reverse-engineered in Sec. 4. Large pools exceeding L2 capacity necessitate frequent DRAM interactions, which trigger the DCP. The DCP erroneously prefetches victim data (which should have been evicted) back into the cache, violating the LRU assumption central to *PPP*'s pruning logic and causing the construction process to fail.

6.3 SLC Eviction Set Construction

To construct eviction sets at the SLC level, we employ a hierarchical methodology adapting the two-tier mechanism proposed in L2CS [44]. Since unprivileged users lack access to physical addresses or slice mappings, direct candidate filtering is infeasible. To overcome this restriction, we integrate the strategy from ZenLeak [37], constructing L2 eviction sets to filter candidates and serve as occupancy sets. This addresses the visibility gap inherent to non-inclusive hierarchies: candidates resident solely in L2 fail to induce SLC conflicts (false negatives) [18, 41]. By forcing these candidates into the SLC, the occupancy sets enable valid conflict testing. This approach progressively identifies eviction sets in the lower cache hierarchy. **Crucially, guided by our characterization of the SLC Victim Buffer (VB) in Sec. 3.3, we enforce a latency threshold of 225 cycles to eliminate VB interference, ensuring the convergence of the construction algorithm.** As illustrated in Fig. 15, the construction process proceeds in three cascaded phases:

Phase 1: L2 Eviction Set Construction. This phase generates N independent L2 eviction sets. We compare a baseline iterative approach (repeating *PPP* or *GE* N times) against an optimized Prune+PlumTree [13] (*PPT*) strategy. *PPT* employs recursive binary partitioning to identify eviction sets with logarithmic complexity ($O(\log N)$), significantly mitigating construction overhead.

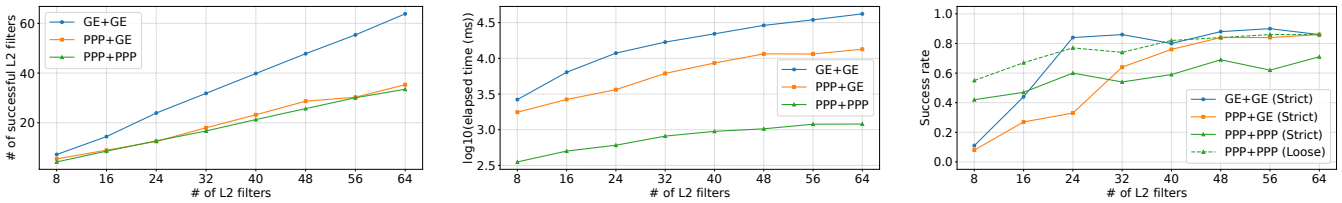
Phase 2: SLC Candidate Filtration. In this phase, the N L2 eviction sets generated in *Phase 1* are repurposed as filters. We utilize these established conflict sets to screen the large-scale address pool, identifying cachelines that are evicted from the L2 into the SLC. These addresses constitute the refined SLC candidate pool.

Phase 3: SLC Eviction Set Reduction. With the search space significantly constrained by *Phase 2*, we apply standard reduction algorithms to eliminate redundant addresses.

To quantify practical efficacy on the Apple M4 platform, we systematically evaluated three algorithm combinations: **GE+GE**, **PPP+GE**, and **PPP+PPP**.

We excluded Prune+PlumTree (*PPT*) [13] despite its theoretical efficiency, as M4 microarchitectural noise (e.g., DCP) causes cascading errors in its recursive probing, resulting in near-zero success. Similarly, we omitted Slice+Slice Baby (*SSB*) [21] because it strictly requires physical addressing and a known slice hash, prerequisites that are inaccessible on the opaque M4 architecture.

Filter Generation Capability (Fig. 14a). *GE+GE* maintains a near-perfect 1:1 yield, confirming GE's robustness. Conversely, combinations using *PPP* in *Phase 1* exhibit yield attenuation (slope < 1), attributed to *PPP*'s susceptibility to noise during initial construction.



(a) L2 filter validity. (b) Time overhead. (c) SLC success rate.

Figure 14: Performance comparison: L2 filter validity, time overhead, and SLC success rate.

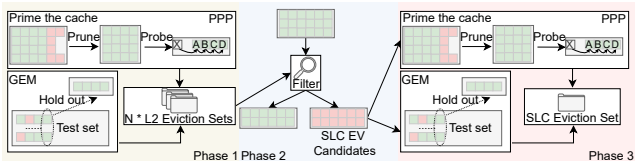


Figure 15: Overview of the hierarchical SLC eviction set construction process.

Time Overhead Analysis (Fig. 14b). *PPP+PPP* demonstrates superior efficiency, completing in approximately 0.3s to 1s. In contrast, *GE+GE* is the most computationally expensive, scaling up to 35s as L2 filters increase, while *PPP+GE* occupies the intermediate range. Overall, execution time correlates positively with the filter count for all strategies

Construction Quality (Fig. 14c). Success scales with filter count (N). *GE+GE* exceeds 80% success at $N \geq 24$, while *PPP+GE* requires $N > 40$. *PPP+PPP* reveals a consistent $< 10\%$ performance gap favoring the *Loose* mode ($< 2W_{SLC}$).

6.4 Characterizing Eviction Set Stability

We stress-tested 50 successful eviction sets over 30 iterations by executing the full eviction sequence (Target + Set). The latency distribution of the set members is visualized in Fig. 16.

L2 Eviction Set Stability: PPP (Fig. 16a). Latencies are highly concentrated within the SLC hit range, confirming robust construction where all members are evicted to SLC.

SLC Eviction Sets: PPP+GE (Fig. 16b). These sets reveal the SLC's *Sticky Allocation* policy, where specific members persistently resist eviction, manifesting as sustained SLC hit latencies (≈200 cycles). We attribute the 14% failure rate directly to this policy: if the target line acquires this sticky state, it becomes immune to standard eviction. Consequently, success relies probabilistically on the target remaining in a non-sticky state.

SLC Eviction Sets: PPP+PPP (Fig. 16c). PPP+PPP yields highly unstable sets characterized by mixed latencies, indicating significant noise. Pruning analysis reveals that removing at most five members from the eviction set causes eviction failure. We conclude that this configuration constructs an *L2 Occupancy Set* rather than a precise SLC eviction set: it relies on secondary contention generated by displacing L2 filter lines into the SLC. This indirect mechanism inherently introduces more noise than minimal eviction sets.

6.5 Characterizing Eviction Set Resolution

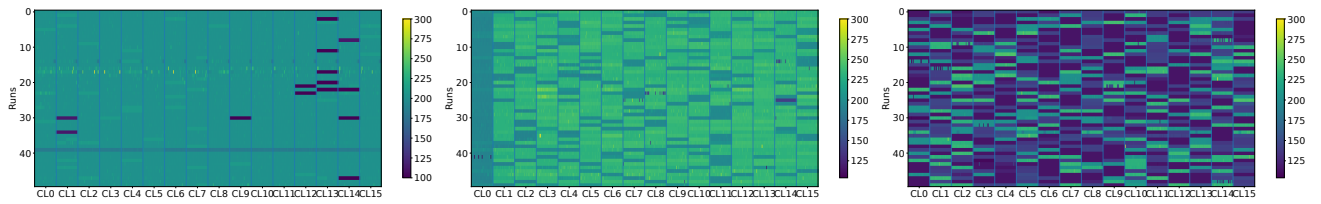
The previous sections focus on whether eviction sets can be constructed and how long the construction procedure takes. In a practical side-channel setting, however, a constructed eviction set is reused repeatedly after construction. We therefore further evaluate the *attack-time resolution* of the resulting eviction primitive, including both temporal resolution and spatial resolution.

Temporal resolution of eviction. Once a valid eviction set is fixed, the attack-time eviction cost is primarily determined by the eviction-set size, the number of access rounds, and the memory-access sequence, rather than by the construction algorithm itself. We therefore directly evaluate the temporal resolution of the constructed L2 and SLC eviction sets. For each constructed set, we repeatedly execute the eviction sequence and measure the cycle count required to complete one attack-time eviction operation.

We evaluate the attack-time temporal resolution of the constructed eviction sets by measuring their resulting access cost. For L2 eviction sets, one eviction operation takes an average of 4658.16 ± 67.24 cycles (95% confidence interval). For SLC eviction sets, the cost depends on the number of L2 filters used during construction, since these filters determine the amount of cache activity required to drive candidates into the SLC and create SLC-level contention. With 8 L2 filters, one SLC eviction operation takes an average of 13421.00 ± 126.11 cycles. Increasing the number of L2 filters to 64 increases the cost to 86011.03 ± 614.96 cycles. We therefore use 40 L2 filters as a moderate configuration, whose mean cost is 51154.31 ± 388.04 cycles.

These results separate the one-time construction cost from the cost of using the eviction set as a repeated monitoring primitive. L2 eviction provides a low-latency monitoring primitive, while SLC eviction incurs higher access cost due to the additional L2-filter activity.

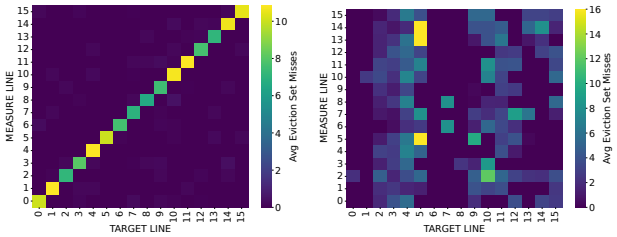
Spatial resolution of eviction. We next evaluate whether the constructed eviction sets can distinguish neighboring cache-



(a) L2 eviction set (*PPP*). (b) SLC eviction set (*PPP+GE*). (c) SLC eviction set (*PPP+PPP*).

Figure 16: Heatmaps of latency distribution for L2 and SLC eviction set members.

line accesses. Since the Apple M4 uses 128-byte cache lines, we evaluate spatial resolution at cache-line granularity. For each measured cache line, we first construct an eviction set targeting that line. We then let the victim access one of several neighboring cache lines and run a Prime+Probe sequence. During the probe phase, we measure the number of misses observed in the eviction set. A larger number of misses indicates that the victim access conflicted with the measured cache line.



(a) Optimized (*PPP*). (b) Minimally adapted (*PPP*).

Figure 17: Spatial-resolution heatmaps for neighboring 128-byte cache-line accesses.

The resulting cacheline confusion matrices are illustrated in Figure 17. The x-axis denotes the victim's target cache line, while the y-axis denotes the measured cache line whose eviction set is probed. Each cell reports the average number of eviction-set misses observed during the probe phase. With our optimized PPP construction, the heatmap exhibits a clear diagonal pattern: accesses to a target cache line mainly increase the miss count of the eviction set constructed for the same cache line. This shows that neighboring cache-line accesses can be clearly distinguished. In contrast, the minimally adapted PPP baseline does not show a stable diagonal structure. Here, "minimally adapted" refers to a PPP implementation in which the system-level parameters required for Apple Silicon are calibrated to the M4 platform, including the timing source, cache-line size, cache-level latency thresholds, and other measurement parameters. This baseline does not include the M4-specific optimizations introduced in this paper. Its diffuse heatmap indicates that the resulting eviction sets do not provide reliable cache-line-level spatial resolution.

AES T-table cache-line classification. We evaluate whether the improved spatial resolution translates to a T-table-style side-channel benchmark. We define accuracy as top-1 cache-line classification accuracy. A trial is counted as correct if the predicted 128-byte T-table cache-line class matches the ground-truth cache-line class accessed by the victim. For each classification trial, we execute 30 Prime+Probe operations, each of which takes approximately 4,000 cycles, excluding the one-time construction cost.

Using the AES T-table benchmark, the minimal adapted PPP algorithm achieves only 4% accuracy. This low accuracy is mainly due to the fact that its constructed sets fail to be reliably evicted by the victim-accessed cache line, resulting in weak and noisy classification signals. In contrast, guided by the reverse-engineered M4 cache behaviors, our optimized PPP construction achieves 78% accuracy under the same Prime+Probe budget. Since both methods use the same number of Prime+Probe operations, the improvement comes from stronger distinguishability rather than additional measurement overhead.

7 Discussion

7.1 Future Work

Our findings on the Apple M4 lay the groundwork for several promising directions in hardware security research:

Optimizing PPP+PPP Construction. To mitigate the instability of *PPP+PPP*, future research should prioritize pinpointing the specific execution phases where non-conflicting L2 lines are inadvertently retained. Characterizing the precise timing of this noise injection will enable targeted filtering mechanisms, refining the current occupancy sets into rigorous, minimal SLC eviction sets and significantly improving construction success rates.

Deterministic State Manipulation. Overcoming *sticky allocation* requires investigating memory operation sequences capable of deterministically resetting the SLC's high-priority state. If an attacker can force a target line out of this privileged retention, the success rate of eviction set construction could theoretically approach 100%.

Methodological Generalizability. While focused on the M4, our core methodology, specifically prefetcher profiling and

IPV analysis, is applicable across the broader ARM64 ecosystem. This is validated on the M2, where we achieved 73.5% (L2 *PPP*), 95% (SLC *GE+GE*), and 78% (SLC *PPP+PPP*, *loose*). Future work involves extending these techniques to characterize other high-performance ARM cores.

End-to-End Cross-Boundary Exploitation. Constructing SLC eviction sets unlocks novel side-channels between heterogeneous components (e.g., P/E clusters, CPU-GPU). This renders end-to-end exploits, previously deemed impractical on Apple Silicon, feasible.

7.2 Countermeasures

Our findings underscore the necessity of hardware defenses.

Randomization and Partitioning. While mitigation strategies like cache set randomization and strict partitioning exist [9, 19, 26, 38], they remain largely absent in commercial processors (including the M-series) due to prohibitive performance overhead and implementation complexity.

Heuristic Detection. A more deployable strategy involves monitoring statistical anomalies characteristic of eviction set construction [31]. Upon detection, the system could trigger a temporary cache flush to nullify the attacker's progress.

8 Conclusion

We resolve eviction set construction impediments on Apple M-chips by characterizing SLC victim buffer, DCP noise, and an opaque SLC allocation policy that is modulated by L2 IPV. Our optimized primitives achieve 88.2% success for L2 Prime-Prune-Probe (PPP), and 90% success for SLC eviction sets. These contributions unlock necessary capabilities for cross-core and cross-component side-channels on Apple Silicon.

ACKNOWLEDGEMENTS

This paper is supported by National Natural Science Foundation of China No. 62361166633 and Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM114).

Ethical Considerations

We follow a stakeholder-based ethics analysis informed by the principles of the Menlo Report, namely Beneficence, Respect for Persons, Justice, and Respect for Law and Public Interest. Our analysis considers both the ethics of conducting this research and the ethics of publishing it.

Stakeholders. The stakeholders in this work include: (1) end users of affected systems, who may be exposed to security risks if the underlying microarchitectural behaviors are abused; (2) hardware and software vendors, particularly Apple, who bear responsibility for investigating, mitigating, and communicating about such risks; (3) the security research community, which benefits from rigorous empirical understanding of microarchitectural attack surfaces and defenses; and (4) the authors and their institutions, who are responsible for conducting the research in a lawful, controlled, and scientifically responsible manner.

Research Procedure and Experimental Safety. All experiments were conducted on hardware owned by the research team in controlled and isolated environments. We did not access third-party devices, accounts, or data, nor did we collect personal or private user information. We did not intentionally compromise production systems or attempt unauthorized access to any external infrastructure. Accordingly, we believe the direct harms arising from the research procedure itself were minimal.

Potential Benefits and Harms. The primary benefit of this work is to improve the understanding of previously undocumented or insufficiently understood microarchitectural behaviors that can affect the security of real systems. Such understanding is important for evaluating the robustness of existing defenses, informing future hardware and software mitigations, and enabling more realistic security assessments.

At the same time, publication creates a non-negligible dual-use risk: attackers may use the insights in this paper to better understand or exploit related microarchitectural weaknesses. This risk applies both to the attack methodology and to the reverse-engineering techniques used to characterize the underlying mechanisms. In addition, disclosure may impose engineering and coordination costs on vendors responsible for mitigation.

Mitigations. We took several steps to reduce these risks. First, we responsibly disclosed our findings to Apple Product Security on February 5, 2026, providing a technical summary, a draft of this paper, and reproduction artifacts. We followed up with Apple on May 21, 2026, informing them that this work was expected to be publicly released no earlier than August 12, 2026, and asking whether Apple requested any embargo, extension, or additional coordination before public release. Apple responded on May 22, 2026, that they had reviewed the disclosure and had no reservations about the research becoming public. Apple did not request an embargo, extension, or additional coordination, and did not indicate any planned mitigations. Second, while adhering to open-science expectations for peer review, we limited artifact release during the active disclosure period: the anonymized artifacts provided for evaluation are scoped to the scientific claims of the paper, such as eviction-set construction algorithms and microarchitectural analysis tools, and do not include weaponized exploit code or a turnkey exploitation pipeline. Third, we confined our evaluation to controlled laboratory settings and avoided experiments on third-party systems or sensitive workloads.

Residual Risk. Some risk remains unmitigated. In particular, even without releasing weaponized artifacts, the paper may still lower the barrier for well-resourced adversaries by revealing security-relevant microarchitectural behaviors and measurement strategies. We cannot completely eliminate this dual-use concern. We therefore intentionally omit operational details that are unnecessary for scientific evaluation but would make immediate misuse easier.

Ethical Analysis and Decision to Proceed. Under a Beneficence analysis, we believe the expected benefits of the work outweigh the residual risks: identifying and documenting security-relevant hardware behaviors is important for enabling scrutiny, mitigation, and more robust system design. Under Respect for Persons, we avoided using devices or data from non-consenting individuals and did not expose users to direct experimental risk. Under Justice, the work aims to benefit the broader ecosystem of users, defenders, and vendors rather than a narrow private interest. Under Respect for Law and Public Interest, we conducted the work on owned equipment, followed responsible disclosure practices, and limited the release of sensitive artifacts during the active coordination period.

Based on these considerations, we concluded that it was ethically appropriate to conduct the research and publish it in a measured manner, provided that responsible disclosure, controlled experimentation, and a scoped artifact release were maintained.

Open Science

We are fully committed to the principles of open science and reproducibility. We will make publicly and permanently available a complete artifact snapshot containing the source code, benchmark programs, eviction-set construction algorithms, reverse-engineering experiments, data-processing scripts, and plotting scripts used in this paper. The artifact is available at:

https://doi.org/10.5281/zenodo.20336892

The artifact is organized by experiment modules. Below, we enumerate the publicly available components and map each artifact directory to the corresponding results in the main body of the paper.

- `01_Benchmark/01_cache_latency`: Contains the experiments and scripts for Fig. 1a and Fig. 2a.
- `01_Benchmark/02_cache_flush_latency`: Contains the experiments and scripts for Fig. 1b and Fig. 2b.
- `01_Benchmark/03_slc_victim_buffer`: Contains the experiments and scripts for Fig. 1c.
- `02_DCP/01_behaviror`: Contains the experiments and scripts used to produce the results shown in Fig. 3.
- `02_DCP/02_minimal`: Contains the experiments and scripts used to produce the results shown in Fig. 4.
- `02_DCP/03_pattern`: Contains the experiments and scripts used to produce the results shown in Fig. 5.
- `02_DCP/04_mitigation`: Contains the experiments and scripts used to produce the results shown in Fig. 6.
- `02_DCP/05_case`: Contains the experiments and scripts used to produce the results discussed in Sec. 4.5.
- `03_IPV/01_PIP`: Contains the experiments and scripts used to produce the results shown in Fig. 8.
- `03_IPV/02_I2P/01_heatmap`: Contains the experiments and scripts used to produce the results shown in Fig. 10.
- `03_IPV/02_I2P/02_transition`: Contains the experiments and scripts used to produce the results shown in Fig. 11 and Table 1.
- `04_EV/01_L2/01_access_frequency`: Contains the experiments and scripts used to produce the results shown in Fig. 13.
- `04_EV/01_L2/02_backtrack`: Contains the experiments and scripts used to produce the results reported in Table 4.
- `04_EV/01_L2/03_candidates`: Contains the experiments and scripts used to produce the results reported in Table 2.
- `04_EV/01_L2/04_random`: Contains the experiments and scripts used to produce the results in Table 3.
- `04_EV/02_SLC`: Contains the experiments and scripts used to produce the results shown in Fig. 14.
- `04_EV/04_sec_eva`: Contains the experiments and scripts used to produce the results discussed in Sec. 6.5.

Together, these artifacts cover the main experimental contributions of the paper, including cache-latency characterization, DCP behavior analysis, IPV-driven replacement-policy experiments, optimized L2/SLC eviction-set construction, and the downstream security evaluation.

References

[1] Apple unleashes M1. https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/.

[2] Google Axion Processors | Google Cloud. https://cloud.google.com/products/axion?hl=en.

[3] Announcing Amazon EC2 M4 and M4 Pro Mac instances | AWS News Blog, September 2025. https://aws.amazon.com/blogs/aws/announcing-amazon-ec2-m4-and-m4-pro-mac-instances/.

[4] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin Von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. A systematic evaluation of transient execution attacks and defenses. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 249–266, 2019.

[5] Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher Fletcher, David Kohlbrenner, Riccardo Paccagnella, and Daniel Genkin. GoFetch: Breaking Constant-Time cryptographic implementations using data Memory-Dependent prefetchers. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1117–1134, 2024.

[6] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. SgxPectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157. IEEE, 2019.

[7] Yun Chen, Ali Hajiabadi, Lingfeng Pei, and Trevor E Carlson. Prefetchx: Cross-core cache-agnostic prefetcher-based side-channel attacks. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 395–408. IEEE, 2024.

[8] Yun Chen, Lingfeng Pei, and Trevor E Carlson. Afterimage: Leaking control flow data and tracking load operations via the hardware prefetcher. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 16–32, 2023.

[9] Lukas Giner, Stefan Steinegger, Antoon Purnal, Maria Eichlseder, Thomas Unterluggauer, Stefan Mangard, and Daniel Gruss. Scatter and split securely: Defeating cache contention and occupancy attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2273–2287. IEEE, 2023.

[10] Sandeep Gupta and Perumal R Subramoniam. Victim allocations in shared system cache, March 30 2021. US Patent 10,963,392.

[11] Lorenz Hetterich and Michael Schwarz. Branch Different-spectre attacks on apple silicon. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 116–135. Springer, 2022.

[12] Tyler J Huberty, Henry A Pellerin, and Daniel A Jimenez. Insertion/promotion vectors to update replacement data in caches based on criticality, February 11 2025. US Patent 12,222,875.

[13] Tom Kessous and Niv Gilboa. Prune+ PlumTree-finding eviction sets at scale. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3754–3772. IEEE, 2024.

[14] Jason Kim, Jalen Chuang, Daniel Genkin, and Yuval Yarom. FLOP: Breaking the apple m3 cpu via false load output predictions. In *USENIX Security Symposium (USENIX Security)*, 2025.

[15] Jason Kim, Daniel Genkin, and Yuval Yarom. SLAP: Data speculation attacks via load address prediction on apple silicon. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3549–3566. IEEE, 2025.

[16] Jason Kim, Stephan Van Schaik, Daniel Genkin, and Yuval Yarom. ileakage: browser-based timerless speculative execution attacks on apple devices. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2038–2052, 2023.

[17] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. *Communications of the ACM*, 63(7):93–101, 2020.

[18] Zili Kou, Sharad Sinha, Wenjian He, and Wei Zhang. Attack directories on arm big. LITTLE processors. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[19] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B Lee. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 IEEE international symposium on high performance computer architecture (HPCA)*, pages 406–418. IEEE, 2016.

[20] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pages 605–622. IEEE, 2015.

[21] Bradley Morgan, Gal Horowitz, Sioli O'Connell, Stephan van Schaik, Chitchanok Chuengsatiansup, Daniel Genkin, Olaf Maennel, Paul Montague, Eyal Ronen, and Yuval Yarom. Slice+Slice Baby: Generating

last-level cache eviction sets in the blink of an eye. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3479–3496. IEEE, 2025.

[22] Paul Nash. Azure Virtual Machines with Ampere Altra Arm–based processors—generally available, August 2022.

[23] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. {DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks. In *25th USENIX security symposium (USENIX security 16)*, pages 565–581, 2016.

[24] Antoon Purnal, Furkan Turan, and Ingrid Verbauwhede. Prime+Scope: Overcoming the observer effect for high-precision cache contention attacks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2906–2920, 2021.

[25] Antoon Purnal and Ingrid Verbauwhede. Advanced profiling for probabilistic Prime+ Probe attacks and covert channels in ScatterCache. *arXiv preprint arXiv:1908.03383*, 2019.

[26] Moinuddin K Qureshi. Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 775–787. IEEE, 2018.

[27] Moinuddin K Qureshi. New attacks and defense for encrypted-address cache. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 360–371, 2019.

[28] Joseph Ravichandran, Weon Taek Na, Jay Lang, and Mengjia Yan. PACMAN: attacking arm pointer authentication with speculative execution. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 685–698, 2022.

[29] Aditya Rohan, Biswabandan Panda, and Prakhar Agarwal. Reverse engineering the stream prefetcher for profit. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 682–687. IEEE, 2020.

[30] Till Schlüter, Amit Choudhari, Lorenz Hetterich, Leon Trampert, Hamed Nemati, Ahmad Ibrahim, Michael Schwarz, Christian Rossow, and Nils Ole Tippenhauer. Fetchbench: Systematic identification and characterization of proprietary prefetchers. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 975–989, 2023.

[31] Wei Song, Boya Li, Zihan Xue, Zhenzhen Li, Wenhao Wang, and Peng Liu. Randomized last-level caches are still vulnerable to cache side-channel attacks! but we can fix it. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 955–969. IEEE, 2021.

[32] Wei Song and Peng Liu. Dynamically finding minimal eviction sets can be quicker than you think for Side-Channel attacks against the LLC. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, pages 427–442, 2019.

[33] Jan Philipp Thoma and Tim Güneysu. Write me and i'll tell you secrets–write-after-write effects on intel cpus. In *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 72–85, 2022.

[34] Jose Rodrigo Sanchez Vicarte, Michael Flanders, Riccardo Paccagnella, Grant Garrett-Grossman, Adam Morrison, Christopher W Fletcher, and David Kohlbrenner. Augury: Using data memory-dependent prefetchers to leak data at rest. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1491–1505. IEEE, 2022.

[35] Pepe Vila, Boris Köpf, and José F Morales. Theory and practice of finding eviction sets. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 39–54. IEEE, 2019.

[36] Alan Wang, Boru Chen, Yingchen Wang, Christopher W Fletcher, Daniel Genkin, David Kohlbrenner, and Riccardo Paccagnella. Peek-a-walk: Leaking secrets via page walk side channels. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3534–3548. IEEE, 2025.

[37] Han Wang, Ming Tang, Quancheng Wang, Ke Xu, and Yinqian Zhang. ZenLeak: Practical last-level cache side-channel attacks on amd zen processors. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pages 1–7. IEEE, 2025.

[38] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. ScatterCache: thwarting cache attacks via cache set randomization. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 675–692, 2019.

[39] Tianhong Xu, Aidong Adam Ding, and Yunsi Fei. EXAM: Exploiting exclusive system-level cache in apple m-series socs for enhanced cache occupancy attacks. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, pages 1294–1308, 2025.

[40] Zihan Xue, Jinchi Han, and Wei Song. CTPP: A fast and stealth algorithm for searching eviction sets on intel processors. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 151–163, 2023.

[41] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 888–904. IEEE, 2019.

[42] Jiyong Yu, Aishani Dutta, Trent Jaeger, David Kohlbrenner, and Christopher W Fletcher. Synchronization storage channels (S2C): Timer-less cache Side-Channel attacks on the apple m1 via hardware synchronization instructions. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1973–1990, 2023.

[43] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316, 2012.

[44] Zirui Neil Zhao, Adam Morrison, Christopher W Fletcher, and Josep Torrellas. Last-level cache side-channel attacks are feasible in the modern public cloud. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 582–600, 2024.