Do Modern LLC Replacement Policies Generalize to Modern Workloads?

AJ Romeo, Luciano Galvani

ABSTRACT

Cache replacement policies and algorithms have been a prominent area of research for several decades, with re-reference prediction approaches such as RRIP and its variants exhibiting strong gains over LRU on standard benchmarks. However, these works have been predominantly evaluated against SPEC CPU workloads which are increasingly diverging from the instruction-footprint-dominated workloads seen in modern hyperscale and data center computing. In this paper, we present a multi-workload evaluation of six LLC replacement policies: LRU, SRRIP, DRRIP, SHiP, Hawkeye, and Mockingjay across three benchmark suites: SPEC CPU 2006, SPEC CPU 2017, and Google Workload Traces v2 using the ChampSim simulator. We find that the traditional policies which outperform LRU on the classic SPEC benchmark suites, and their variants, fail to generalize to the Google Workload Traces, where the same policies degrade IPC by up to 14% at a 2MiB LLC and converge toward LRU performance as LLC size grows. These results show that the SPEC suite is a poor indicator for evaluating replacement policies in the modern server and cloud computing contexts and suggest that new cache replacement paradigms may be necessary to adapt to the contemporary environment. We attribute this failure to two workload properties: high PC entropy across GWT traces, which renders PC-based signature prediction unreliable, and a dominant IPC bottleneck from branch misprediction rather than LLC miss latency, which replacement policy alone is unlikely to address.

1. INTRODUCTION

Due to the growing disparity between the speed of modern processors and DRAM latency, computer architects have developed cache hierarchies with the last level cache (LLC) acting as the final buffer before main memory. However, the LLC often suffers from poor access patterns due to higher level caches absorbing high-reuse lines, leaving references with poor spatial and temporal locality to fall to the LLC. Modern replacement policies attempt to offset the memory latency by predicting the reuse of cache lines. They use techniques such as re-reference interval prediction [1] and program-counter-indexed sampling [2] to select lines with poor reuse profiles for eviction. Modern policies like the RRIP family, Hawkeye [3], and Mockingjay [4] have shown significant IPC improvements over LRU when evaluated on SPEC CPU 2006 [5] and SPEC CPU 2017 [6] workloads, making them strong candidates for general-purpose computing. However, the SPEC CPU suites are dominated by workloads with predictable, loop-driven access patterns. The rise of data centers and hyperscale computing introduces a new workload domain with sparse, data-dependent traversals and irregular memory accesses with little to no exploitable reuse structure. State-of-the-art policies exploit the predictable, loop-based structure that the SPEC workloads provide, and when they are evaluated on traces representative of these new workloads, they fail to provide the same meaningful performance increases and in some cases degrade performance [7]. In this paper we evaluate six LLC replacement policies (LRU, SRRIP, DRRIP, SHiP, Hawkeye, and Mockingjay) across SPEC CPU 2006, SPEC CPU 2017, and Google Workload Traces v2 (GWT) [8], simulated using ChampSim [9]. Our results indicate that reliance on the SPEC suite as a primary evaluation tool fits cache eviction policies to a narrow class of workloads and does not account for the irregular accesses seen in data center workloads.

2. BACKGROUND

2.1 LLC Replacement Fundamentals

Replacement policies differ primarily in what signal they use to predict future reuse. Recency-based policies assume that recently accessed lines are more likely to be reused. Re-reference prediction policies maintain explicit estimates of future reuse distance. Signature-based policies use program context, typically the program counter, to learn whether lines generated by similar instructions tend to be reused. OPTgen-based policies use sampled access histories to approximate Belady's optimal policy and train predictors from that approximation.

These approaches are effective when the workload provides stable reuse signals. However, LLC accesses are often the residual stream left after upper-level caches have already absorbed high-locality references. For irregular workloads, especially those with sparse or data-dependent memory behavior, these reuse signals may become weaker and noisier. In this setting, increasingly sophisticated replacement policies may lose their advantage over simpler baselines in the absence of clear predictive patterns.

2.2 RRIP and Its Variants

Each cache line is assigned an M-bit Re-Reference Interval Prediction Value (RRPV) at insertion [1]. Greater RRPV values represent more distant predicted reuse, while a value of 0 represents immediate predicted reuse. RRIP variants differ primarily in how they initialize RRPVs on insertion and how they adapt those insertion decisions. During victim selection, these values are aged in response to cache activity and the line with the greatest RRPV is evicted.

2.3 Signature-Based Hit Prediction

Signature-based approaches attempt to improve RRIP insertion decisions by using program-context signatures, typically truncated program counters, to predict whether newly inserted cache lines are likely to be reused [2]. These policies maintain a history table indexed by signature, with saturating counters adjusted based on the observed reuse behavior of lines associated with each signature. Initial reuse predictions are formed from this table and used to select a line's insertion priority.

2.4 OPTgen-Based Policies

OPTgen-based policies move beyond simple insertion heuristics by attempting to approximate Belady's optimal replacement policy using past access behavior [3, 4]. By reconstructing which past accesses would have been cache hits under an optimal policy, these methods train predictors that estimate the future usefulness of newly inserted lines. Specific implementations vary in how much access history they store and whether they classify lines in a binary (useful/not useful) or numeric fashion.

3. POLICIES

3.1 LRU

LRU maintains a recency ordering of cache lines, so recently accessed lines are treated as more likely to be reused. This implementation stores one timestamp per way per set in an array. On each cache fill, the filled line is assigned the current cycle value and the cycle counter is incremented. On each cache hit, the accessed line is updated in the same way, except that writeback hits do not modify the replacement state. Victim selection scans the entries for the current set and returns the way with the minimum timestamp (i.e. the line whose last access was furthest in the past). This policy assumes that the least recently used line is the least likely to be reused, which works well when recent access history is a good predictor of future reuse.

3.2 SRRIP

SRRIP replaces LRU's recency timestamps with a Re-Reference Prediction Value (RRPV) for each cache line. With M-bit RRPVs, values range from 0 (low reuse distance) to $2^M - 1$ (high reuse distance). On insertion, lines receive a fixed RRPV of $2^M - 2$. On a cache hit, the accessed line's RRPV is reset to 0. During victim selection, SRRIP chooses a line with RRPV $2^M - 1$; if none exists, all RRPVs in the set are incremented until at least one line reaches this value. SRRIP assumes that most inserted lines have a high reuse distance, while still giving them a short opportunity to be reused before eviction.

3.3 DRRIP

DRRIP dynamically selects between two RRIP insertion policies, SRRIP and BRRIP, to reduce cache pollution from scan-heavy access patterns. Bimodal RRIP (BRRIP) is a pessimistic policy assuming a scan-heavy workload, inserting lines with RRPV of $2^M - 1$ at high probability and $2^M - 2$ with low probability. In this implementation, this occasional insertion occurs every BIP_MAX^{th} insertion.

DRRIP uses set dueling [**?**] between these policies, tracking their performance across sampled "leader" sets. A per-CPU saturating counter, PSEL, tracks which leader policy sees fewer misses: misses in the BIP leader decrement PSEL, while misses in the SRRIP leader increment PSEL. All follower sets adopt the policy indicated by PSEL, using BRRIP-style insertion when PSEL is greater than half of its maximum value and SRRIP insertion otherwise. Ideally, DRRIP utilizes SRRIP in predictable, locality-heavy environments and switches to BRRIP-style insertion when scanning behavior would otherwise thrash the cache.

3.4 SHiP

SHiP associates each cache line with a signature at insertion time based on the program counter (PC). The signature history counter table (SHCT) maps each signature to a saturating counter that tracks reuse behavior. In this implementation, the SHCT is trained through sampled sets, where counters are decremented on sampler hits and incremented when a sampled entry that was previously reused is replaced. If a cache line's signature holds the max SHCT value, then it is inserted with an initial RRPV of $2^M - 1$, making it immediately eligible for eviction; otherwise, it is inserted at $2^M - 2$. After insertion, SHiP employs the same hit-promotion, RRPV aging, and victim-selection mechanics as SRRIP. The SHiP algorithm relies on the correlation between a line's reuse behavior and the PC that generated the access [2].

3.5 Hawkeye

Hawkeye is an OPTgen-based replacement policy that approximates Belady's optimal replacement decisions by classifying lines as either cache-friendly or cache-averse [3]. In this implementation, optimal cache behavior is reconstructed for sampled sets. These results then train separate demand and prefetch predictors indexed by PC-based signatures.

On insertion, Hawkeye queries the appropriate predictor for the incoming line's PC-based signature. Cache-averse lines are inserted with maximum RRPV, while cache-friendly lines are inserted with RRPV 0. Similar to other RRIP-based policies, lines are set to RRPV 0 on cache hits and cache-averse lines with maximum RRPV are evicted first. However, if no such line exists, it evicts the line with the largest RRPV among the remaining cache-friendly candidates rather than aging the set as SRRIP does.

It is important to note that although Hawkeye tends to outperform simpler policies on standard workloads, it comes at the cost of substantial metadata. The implementation maintains per-set OPTgen state, sampled address histories, PC signatures, prefetch metadata, and separate demand and prefetch predictors, rendering it less practical for constrained hardware implementations.

3.6 Mockingjay

Mockingjay predicts reuse distance in its approximation of Belady's algorithm rather than making a binary useful/not-

Table 1: Simulation configuration.

Component	Configuration
Core	single core, out-of-order
ROB	352 entries
Branch predictor	Bimodal
L1I cache	32 KiB, 8-way
L1D cache	48 KiB, 12-way
L2 cache	512 KiB, 8-way
LLC	2 MiB, 16-way
LLC prefetcher	None

useful classification [4]. In the ChampSim implementation, each cache line stores an Estimated Time Remaining (ETR) value, while a reuse-distance predictor (RDP) maps hashed PC-based signatures to predicted reuse distances. This combines RRIP-style per-line replacement state with signature-based prediction.

The predictor is trained using sampled cache sets. Sampled hits provide reuse-distance observations, while sampled lines that age beyond the maximum reuse distance penalize their signatures by increasing the predicted reuse distance. During victim selection, Mockingjay evicts the line with the largest absolute ETR value, breaking ties in favor of negative ETR values. The policy also estimates the reuse distance of the incoming line, potentially bypassing it if that prediction is too large. In this way, Mockingjay attempts to proactively preserve lines with shorter predicted reuse distances.

4. METHODOLOGY

4.1 Simulation Infrastructure

We use ChampSim [9] as our microarchitectural simulator.

We evaluate six policies: LRU, SRRIP, DRRIP, SHiP, Hawkeye, and Mockingjay using the ChampSim trace-based architectural simulator. All simulations are run with a single-core configuration with a 1M instruction warmup period, followed by a 100M instruction simulation period. The three level cache hierarchy is illustrated in Table 1. We used a 2MiB LLC for all simulations unless otherwise specified. Policy implementations were derived from their original specifications; parameters are listed in Table 2.

Our evaluation spans three benchmark suites covering a range of application domains. The SPEC CPU 2006 [5] and 2017 [6] suites provide our baseline used in prior cache replacement policy works. The Google Workload Traces [8] represent a set of real data center workloads captured from Google's infrastructure. Traces are grouped by application domain, allowing for isolation of policy behavior across workload classes. The characteristics of each suite are discussed in Section 5.

The primary metrics evaluated are instructions per cycle (IPC) and misses per kilo-instruction (MPKI). IPC captures the overall performance impact of replacement decisions, while MPKI measures the policy's effectiveness independent from other pipeline effects. IPC is normalized to the LRU baseline and given as the geometric mean, so as to give a more robust measure when comparing across varied workloads.

Table 2: Policy parameter configurations used in all simulations.

Policy	Parameter	Value
SRRIP	RRPV bits	2
	Insertion RRPV	2
DRRIP	RRPV bits	2
	SDM leader sets	32
	PSEL counter width	10 bits
	BIP throttle	32
SHIP	RRPV bits	2
	SHCT entries	16,384
	SHCT counter max	7 (3-bit)
HAWKEYE	RRPV bits	3
	Sampled cache size	2,800 lines
	Sampler ways	8
	Predictor size	2,048 entries
MOCKINGJAY	Sampled cache ways	5
	Sampled cache sets	16
	History window	8 intervals
	Timestamp bits	8

5. WORKLOAD CHARACTERIZATION

We characterize the memory behavior of all benchmark suites under an LRU baseline with an 8MiB LLC. 42 traces are selected (13 SPEC CPU 2006 workloads, 14 SPEC CPU 2017, 15 GWT) and analyzed using an instrumented implementation of LRU that tracks approximate reuse distance distributions, per-PC hit/miss rate for each benchmark, and PC entropy, along with LLC MPKI and IPC metrics from ChampSim. We use PC entropy as a measurement of how many unique load program counter values are generating LLC accesses.

5.1 LLC Miss Rate

Figure 1 shows the LLC miss-per-kilo-instruction (MPKI) for all benchmarks under LRU at 8 MiB. Within SPEC CPU 2017, mcf_s peaks at 85 MPKI due to its pointer-chasing graph traversal structure. lbm_s and fotonik3d follow with MPKI around 30 and 18 respectively, reflecting large instruction footprints. Within SPEC CPU 2006, the fp_scientific workloads (milc, GemsFDTD, and bwaves) also show high MPKI from large, strided accesses.

The GWT workloads fall between 2 and 16 MPKI, with delta and charlie exhibiting the highest values, reflecting the pointer-chasing patterns of data center software.

The high MPKI of mcf_s relative to the GWT workloads was a surprising finding since both exhibit irregular memory access patterns. Table 3 shows that mcf_s generates far more LLC accesses ($\sim$ 22% of instructions) missing on 8.6% of instructions, whereas GWT benchmarks such as delta generate LLC accesses on only 3.4% of instructions. The difference in MPKI arises from a difference in frequency of LLC accesses and a fundamentally different instruction mix. mcf_s's

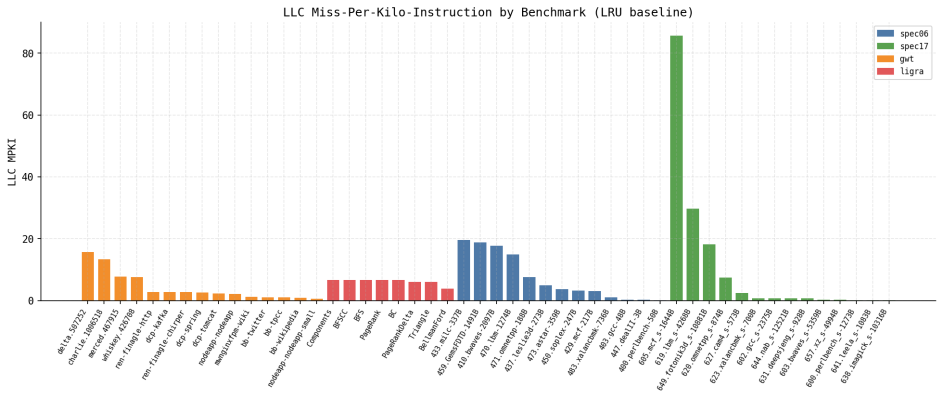

Figure 1: LLC MPKI by benchmark under LRU at 8 MB.

Table 3: LLC access and miss rates per instruction for selected benchmarks under LRU at 8 MB LLC. mcf_s generates LLC accesses at 6× the rate of delta despite both exhibiting irregular access patterns.

Metric	mcf_s	lbm	delta
Suite	S'17	S'06	GWT
IPC	0.17	0.89	0.80
LLC MPKI	85.72	14.87	15.75
Acc. per Instr.	0.219	0.050	0.034
Miss / Access	39.2%	29.5%	46.4%

instruction stream consists of more memory accesses with less arithmetic, while GWT workloads mix non-memory operations that dilute the per-instruction LLC access rate. The MPKI metric should therefore be viewed as a function of both cache behavior and compute intensity.

5.2 IPC Bottlenecks: Memory Latency vs. Branch Misprediction

A notable feature of Table 4 is that GWT workloads exhibit lower IPC than many high-MPKI SPEC workloads despite having substantially lower LLC MPKI. dacapo-tomcat (IPC 0.78, LLC MPKI 2.24), renaissance-finagle-chirper (IPC 0.74, LLC MPKI 2.73), and renaissance-finagle-http (IPC 0.72, LLC MPKI 2.81) all underperform SPEC workloads with far higher miss rates such as bwaves (IPC 1.10, LLC MPKI 17.77). This inversion is explained by the branch misprediction rate, also shown in Table 4.

GWT JVM and web workloads show branch MPKI of 10–24, but by contrast, SPEC fp_scientific benchmarks have near-zero branch MPKI (lbm at 0.08, GemsFDTD at 0.11, bwaves at 4.53) because their access patterns are dominated by regular loop structures with predictable control flow. The IPC of GWT workloads is therefore suppressed primarily by stalls from branch mispredictions, not by LLC miss latency.

5.3 PC Miss Entropy and Signature-Based Policies

Signature-based policies such as Mockingjay and SHiP predict future cache behavior using the load PC as an indicator for access pattern. Their effectiveness depends on whether LLC misses are concentrated at a small number of load PCs or are spread across many PCs. We measure entropy over the per-PC miss distribution, where 0 indicates a single PC dominates all misses and 1 indicates a uniform distribution.

Table 4: Per-suite characterization under LRU at 8 MB LLC.

Metric	SPEC'06	SPEC'17	GWT
Mean IPC	1.24	1.57	0.97
LLC MPKI	5.36	12.44	5.11
Branch MPKI	6.44	9.89	16.50
PC Entropy	0.487	0.449	0.590

Table 4 shows that GWT workloads consistently exhibit the highest PC miss entropy (mean 0.59), compared with 0.49 for SPEC CPU 2006 and 0.45 for SPEC CPU 2017. 11 of 15 GWT benchmarks exceed the 0.55 entropy threshold, compared with 5 of 13 for SPEC CPU 2006 and 4 of 14 for SPEC CPU 2017. LLC misses originate from many different PCs, making per-PC signatures unreliable predictors.

This implies that a perfect LLC replacement policy would not close the IPC gap between GWT and SPEC workloads because the dominant bottleneck is not LLC miss rate. Therefore evaluation based exclusively on SPEC workloads does not measure performance in a way that is representative of branch-bound workloads that dominate modern data center applications.

6. Results

6.1 IPC Comparison

Figure 2 shows geomean IPC normalized to LRU for all six policies across SPEC CPU 2006 traces at 2MiB LLC. All non-LRU policies improve over the baseline, with Mockingjay and Hawkeye demonstrating the largest performance gains. This is consistent with prior evaluations of these policies on SPEC workloads.

Figure 3 shows the same comparison on the Google Workload Traces. All non-LRU policies, with the exception of Hawkeye, degrade IPC at 2 MiB, with the worst performing policy, SRRIP, reaching about 14% below the LRU geomean. No policy notably outperforms LRU outright on GWT at any

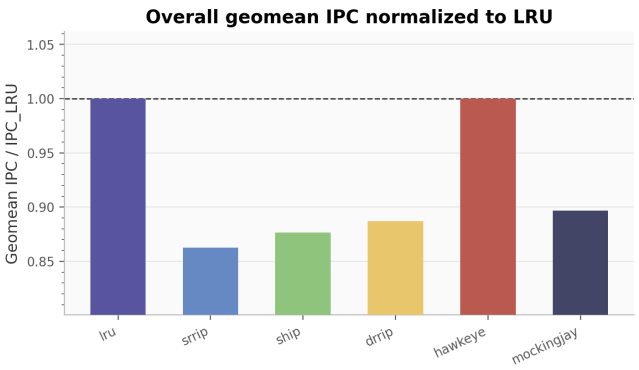

Figure 3: Normalized IPC relative to LRU across GWT traces.

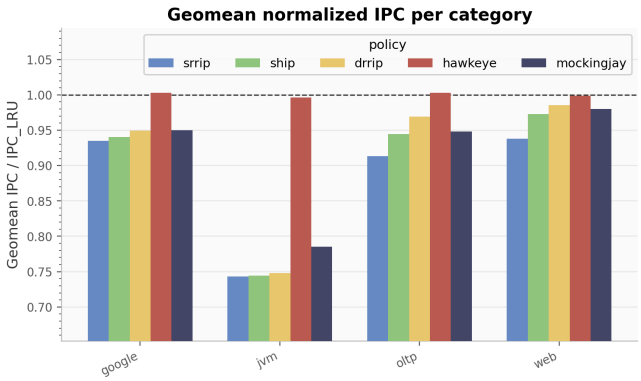

Figure 4: Normalized IPC relative to LRU across GWT traces sorted by workload category.

LLC size tested.

Figure 4 breaks down normalized IPC by GWT workload category. The degradation is consistent across all four subcategories, confirming that the aggregate geomean is not driven by outlier traces. JVM workloads suffer the most severe penalties, with SRRIP, SHiP, and DRRIP reaching approximately 0.74–0.75 and Mockingjay at 0.79, likely reflecting the particularly irregular and data-dependent memory access patterns of these workloads. OLTP and web workloads show milder degradation of 3–9% below LRU, suggesting these categories retain slightly more exploitable locality structure. Hawkeye is the sole policy that remains at or above the LRU baseline across all four categories, a result not shared with Mockingjay despite both being OPTgen-based. This per-category divergence between the two OPTgen policies is examined further in Section 7.

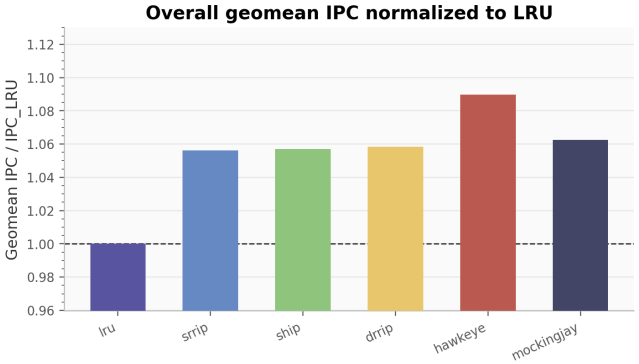

Figure 2: Normalized IPC relative to LRU across SPEC CPU 2006 traces.

Figure 5–7 show normalized geomean IPC as a function of LLC size from 2 MiB to 64 MiB for SPEC CPU 2006, SPEC CPU 2017, and GWT respectively. On SPEC CPU 2006 (Figure 5), policy gains are largest at 2 MiB and shrink as LLC size grows, consistent with working sets fitting in the cache and eviction decisions becoming less consequential.

On SPEC CPU 2017 (Figure 6), differences across policies are smaller and less consistent than on SPEC CPU 2006, with gains confined to a handful of benchmarks. On GWT (Figure 7), all non-LRU policies begin below 1.0 at 2 MiB and converge upward toward LRU as LLC size increases, but never exceed it in aggregate. The convergence pattern confirms that the IPC degradation is a capacity effect: at small LLC sizes, the non-LRU policies' insertion heuristics misclassify GWT lines and pollute the cache. As the LLC grows large enough to absorb more of the working set, eviction decisions matter less and all policies approach equivalent performance.

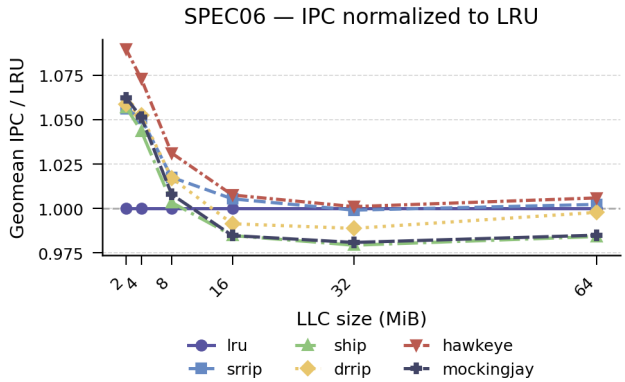

Figure 5: Normalized IPC relative to LRU across LLC sizes on SPEC CPU 2006 traces.

6.2 LLC Miss Rate (MPKI)

Figures 8–10 show geomean LLC MPKI vs. LLC size for SPEC CPU 2006, SPEC CPU 2017, and GWT respectively.

On SPEC CPU 2006 (Figure 8), non-LRU policies show slightly higher MPKI than LRU at 2 MiB before converging by 8 MiB. This is a notable result, as the IPC gains do not arise from reducing total miss count, but from retaining high-reuse lines.

On SPEC CPU 2017 (Figure 9), all policies are nearly indistinguishable across all LLC sizes, consistent with the small and inconsistent IPC differences observed on that suite.

On GWT (Figure 10), the divergence is most pronounced. At 2 MiB, non-LRU policies reach 9.3–10.1 MPKI versus LRU at 5.6 MPKI, an increase of up to 80% in miss rate. All policies converge by 8 MiB and remain indistinguishable

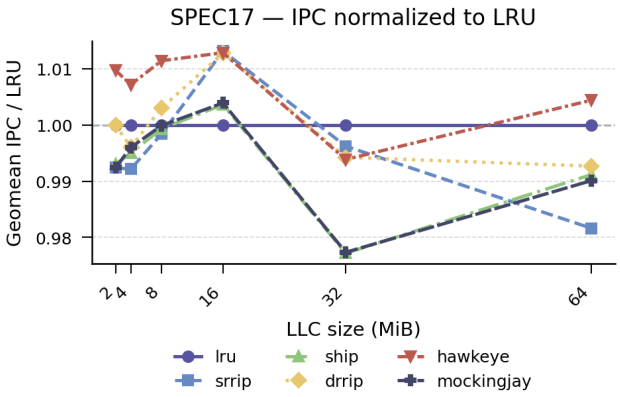

Figure 6: Normalized IPC relative to LRU across LLC sizes on SPEC CPU 2017 traces.

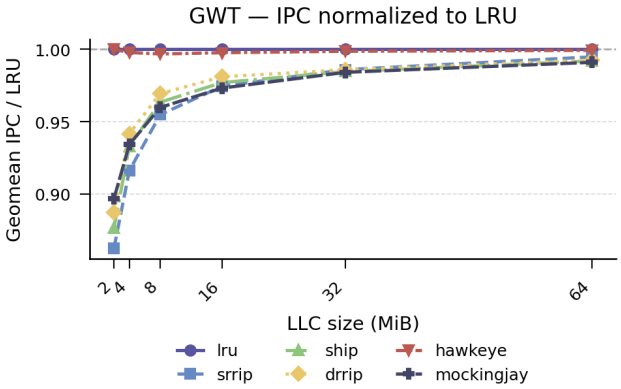

Figure 7: Normalized IPC relative to LRU across LLC sizes on GWT traces.

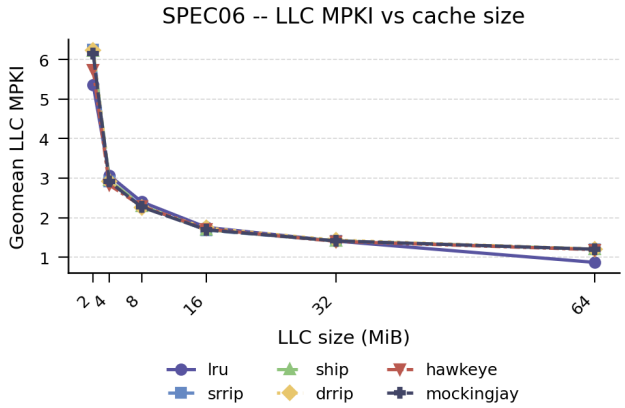

Figure 8: Geomean LLC MPKI vs. LLC size on SPEC CPU 2006.

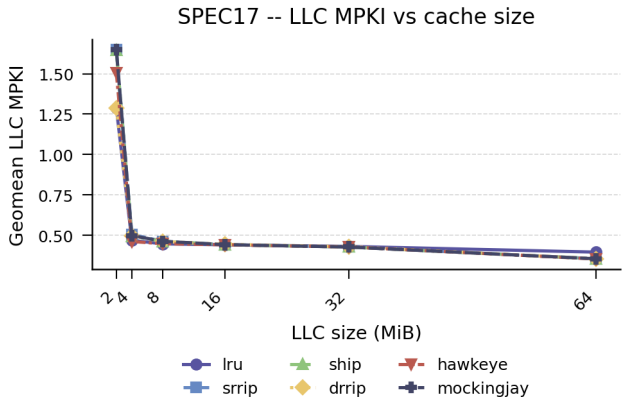

Figure 9: Geomean LLC MPKI vs. LLC size on SPEC CPU 2017.

at larger sizes. This confirms that at small LLC sizes, the insertion heuristics of SRRIP, DRRIP, SHiP, Hawkeye, and Mockingjay actively misclassify GWT lines, evicting lines that should be retained. Combined with the branch-bound IPC bottleneck identified in Section 5.2, this means non-LRU policies impose both a miss rate penalty and fail to address the dominant performance bottleneck on GWT workloads.

7. Discussion

7.1 Why OPTgen-Based Policies May Underperform on GWT

Hawkeye and Mockingjay take their decisions from OPTgen, a component of simulating Belady's optimal algorithm, by replaying the access history within a set to determine which lines would have been hits under OPT. This approach is effective when accesses are loop-structured because the sampled history is representative of future accesses. A line that was a cache hit in the last interval is likely to be a hit in a future interval. On SPEC CPU 2006 and 2017, this assumption holds and the OPTgen-based policies achieve the largest IPC gains of all evaluated policies. On GWT, the assumption breaks down. First, the high PC entropy measured in Section 5.3 means that LLC misses are spread across many PCs rather than concentrated at a few. The sets sampled by Hawkeye and Mockingjay therefore cannot accumulate enough per-PC observations to produce reliable OPTgen classifications. Second, the data-dependent pointer chasing in GWT workloads means that access patterns are not stationary across intervals, and the set of high-reuse lines shifts in ways a fixed window cannot track. The result is that lines predicted as cache-friendly are evicted and lines with little to no reuse are retained. As shown in Figure 4, Mockingjay suffers more severely from this breakdown than Hawkeye across all four GWT subcategories. Mockingjay uses OPTgen output to derive a continuous reuse distance estimate and insert lines at a corresponding RRPV, so noisy classifications translate directly into systematic misplacement. Hawkeye's binary cache-friendly/cache-averse predictor presents a coarser interface to the same noisy signal, which appears to limit the damage on irregular workloads, though a full characterization of this divergence is left to future work.

SRRIP and DRRIP performance degrades for a separate reason. Both policies insert new lines at a high RRPV, assuming most will have low reuse. On SPEC workloads this works well

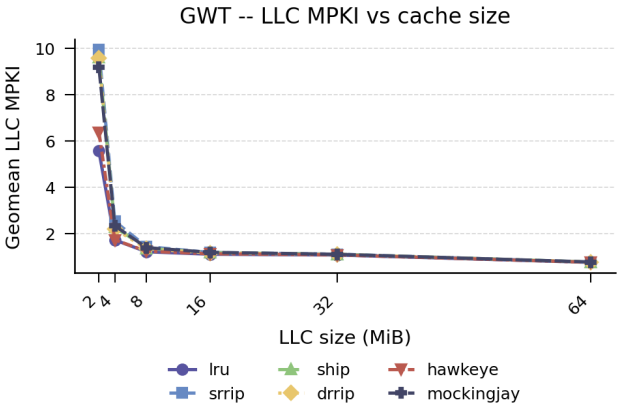

Figure 10: Geomean LLC MPKI vs. LLC size on GWT.

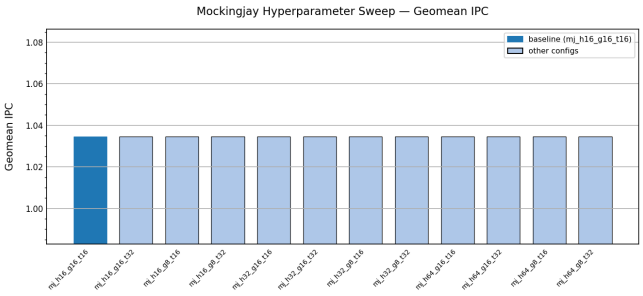

Figure 11: Mockingjay parameter sweep showing no change in IPC. The h value represents history length, g represents granularity, and t represents temporal-difference update rate.

because lines that are reused quickly have their RRPV reset to 0. On GWT, the irregular access patterns mean that many lines are not reused before their RRPV value saturates. DRRIP's set dueling cannot correct this because both SRRIP and BRRIP act similarly under workloads with no exploitable reuse structure. SHiP's PC-based signature mechanism similarly fails when exposed to the high PC miss entropy in the GWT workloads. LRU, by contrast, makes no predictions, simply retaining the lines that were accessed most recently. On workloads with no exploitable reuse structure, this approach is no worse than any prediction-based policy. This is illustrated in Figure 10: as the LLC grows, eviction decisions matter less and the penalty from cache line misclassification diminishes.

7.2 Parameter Sweeping Experiments

After observing that Hawkeye's OPTgen-based benefits did not translate to Mockingjay on the GWT workloads, we suspected that our Mockingjay implementation may have been insufficiently tuned for irregular access patterns. To test this hypothesis, we generated 12 Mockingjay variants by sweeping three core parameters: history length, granularity, and temporal-difference update rate. Specifically, we tested history lengths of 16, 32, and 64; granularities of 8 and 16; and temporal-difference denominators of 16 and 32. For comparison, the standard implementation used a history length of 8, a granularity of 8, and a temporal-difference denominator of 16.

These adjustments were chosen to give Mockingjay more capacity for nuanced reuse-distance prediction in less predictable settings. The history length controls the maximum reuse-distance window represented by the policy, the granularity controls how reuse-distance predictions are compressed into estimated time remaining values, and the temporal-difference rate controls how aggressively the predictor adapts to new samples.

Remarkably, all 12 variants produced the exact same geomean IPC across the evaluated workloads; the results are shown in Figure 11. This suggests that the observed gap between Hawkeye and Mockingjay, and Mockingjay's broader underperformance on GWT, was not merely the result of an unlucky parameter choice. Instead, the sweep indicates that either the implementation is insensitive to these particular parameter changes under the tested workloads, or the GWT traces do not expose reuse patterns that Mockingjay can exploit more effectively through parameter tuning alone.

7.3 Implications for Policy Design

The results reveal two distinct problems. The first is that existing non-LRU policies degrade performance on GWT at small LLC sizes. The second is that even a perfect replacement policy would not close the IPC gap between GWT and SPEC workloads. Section 5.2 shows that GWT workloads are bottlenecked by branch misprediction, not LLC miss latency, a result that no cache replacement algorithm can address. The first is a property of how current policies behave on irregular workloads, and the second is a property of the workloads themselves.

The practical implication is not that replacement policies should be redesigned for larger LLC sizes, where all policies converge to equivalent performance. It is that SPEC CPU is a biased evaluation target. Workloads with loop-driven, PC-predictable access patterns select for policies that exploit that structure, and the gains they produce do not transfer. A policy that scores well on SPEC may still impose a miss rate penalty on the access patterns that dominate data center workloads. The Mockingjay parameter sweep reinforces this point: even when the history window, reuse-distance granularity, and temporal-difference update rate were varied, performance remained unchanged, suggesting that the limitation is not simply a matter of tuning existing reuse-distance predictors. Evaluation frameworks that treat SPEC performance as a proxy for general-purpose cache effectiveness will continue to produce this outcome.

8. Related Work

8.1 LLC replacement policies

The RRIP family of policies [1] introduced the re-reference prediction value abstraction, replacing the binary recency ordering of LRU with a multi-bit reuse distance prediction. SHiP [2] extended this with PC-indexed signatures to improve insertion decisions. Hawkeye [3] and Mockingjay [4]

subsequently introduced OPTgen-based approaches that approximate Belady's optimal algorithm using sampled access history. All of these policies were primarily evaluated on SPEC CPU workloads; our work examines whether their gains transfer to data center traces. PACIPV [10] is a recent policy targeting instruction-heavy workloads, reflecting growing recognition that SPEC-centric evaluation may miss important workload classes.

8.2 Hyperscale workload characterization

Ferdman *et al.* [11] demonstrated through the CloudSuite benchmark suite that scale-out workloads exhibit fundamentally different memory behavior from SPEC, with larger instruction footprints, poor cache utilization, and sensitivity to memory latency that SPEC workloads do not capture. Our results corroborate and extend this finding in the context of LLC replacement policy evaluation. Jamet *et al.* [7] characterize the impact of replacement policies on emerging workloads, finding inconsistent gains outside SPEC. Our work differs in evaluating a broader set of policies across the Google Workload Traces with an LLC size sweep, and in identifying branch misprediction rather than LLC miss rate as a potential IPC bottleneck for data center workloads.

8.3 Benchmark suites

SPEC CPU 2006 [5] and SPEC CPU 2017 [6] remain the standard evaluation targets for cache replacement research. The Google Workload Traces v2 [8] provide representative data center traces captured from Google's production infrastructure, and are the primary non-SPEC suite used in this work.

9. Conclusion

We evaluated six replacement policies, LRU, SRRIP, DRRIP, SHiP, Hawkeye, and Mockingjay, across SPEC CPU 2006, SPEC CPU 2017, and Google Workload Traces using ChampSim. On SPEC CPU 2006, all non-LRU policies outperform the baseline at small LLC sizes, consistent with prior work. On GWT, the same policies degrade IPC at 2 MiB (with SRRIP reaching 14% below LRU) and increase LLC MPKI by up to 80%, converging toward LRU only as the cache grows large enough that eviction decisions become irrelevant. Workload characterization reveals the cause of this failure. GWT workloads exhibit substantially higher PC entropy than either SPEC suite, making PC-based signatures unreliable. Their dominant IPC bottleneck is branch misprediction rather than LLC miss latency, meaning that replacement policy alone is unlikely to close the performance gap. Together, these findings show that the SPEC suites select for a narrow class of locality structure that is increasingly unrepresentative of the workloads running in modern data centers. Policies optimized against SPEC do not generalize, and evaluation frameworks that rely exclusively on SPEC cannot detect this failure mode.

Future work should investigate replacement paradigms that go beyond retuning existing reuse-distance and PC-signature predictors, including mechanisms that detect low-confidence or low-locality regimes and fall back to simpler behavior, while evaluating against benchmark suites that include representative data center traces from the outset.

References

[1] A. Jaleel, K. B. Theobald, J. Steely, Simon C., and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, pp. 60–71, ACM, 2010.

[2] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, J. Steely, Simon C., and J. Emer, "SHiP: Signature-based hit predictor for high performance caching," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 430–441, ACM, 2011.

[3] A. Jain and C. Lin, "Back to the future: Leveraging Belady's algorithm for improved cache replacement," in *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 78–89, IEEE, 2016.

[4] I. Shah, A. Jain, and C. Lin, "Effective mimicry of Belady's MIN policy," in *Proceedings of the 28th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 558–572, IEEE, 2022.

[5] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[6] J. Bucek, K.-D. Lange, and J. v. Kistowski, "Spec cpu2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pp. 41–42, 2018.

[7] A. V. Jamet, L. Alvarez, D. A. Jiménez, and M. Casas, "Characterizing the impact of last-level cache replacement policies on emerging workloads," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, 2020.

[8] "Google workload traces version 2." https://console.cloud.google.com/storage/browser/external-traces-v2.

[9] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," 2022.

[10] S. Mostofi, S. Gupta, A. Hassani, K. Tibrewala, E. Teran, P. V. Gratz, and D. A. Jiménez, "Light-weight cache replacement for instruction heavy workloads," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*, (Tokyo, Japan), pp. 1–15, ACM, 2025.

[11] M. Ferdman, A. Adileh, Y. O. Koçberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 37–48, 2012.