Retrofitting Contextual Learning into Set Dueling

Diamantis Patsidis[§]
diamantis.patsidis2@h-partners.com

Leeor Peled[‡]
leeor.peled@huawei.com

Georgios Vavouliotis[§]
georgios.vavouliotis@huawei.com

[§]Computing Systems Lab, Huawei Technologies AG, Switzerland [‡]Boole Lab, Huawei Tel-Aviv Research Center

***Abstract*—Dynamic policy arbitration is fundamental to modern CPUs, as no single policy performs optimally across diverse workload domains. Set Dueling (SD) is widely used, both in academia and industry, to select between competing policies at runtime, but relies on a global, context-oblivious counter to make decisions, ignoring richer correlations based on execution context such as control and data flow information.**

This work proposes *Context-Aware Set Dueling (CASD)*, a framework that elevates SD to a context-aware, policy-agnostic mechanism for fine-grained policy arbitration at runtime. CASD retrofits contextual learning into context-oblivious SD: a lightweight predictive logic learns correlations between execution context and policy effectiveness, anticipating recurring behavior rather than reacting to slow global counter shifts. This allows accurate, fine-grained adaptation even within a single execution phase, while preserving SD's low-overhead training methodology. More broadly, CASD transforms any microarchitectural scheme that uses context-oblivious policy arbitration into a context-aware one, without modifying the underlying policies.

To demonstrate the benefits of CASD, we design *DuelCeptron*, a CASD prototype that uses perceptron learning to correlate contextual information with policy effectiveness. We evaluate DuelCeptron on three SD applications, showing that it consistently outperforms conventional SD with minimal storage overhead across a large and diverse set of workloads. For example, in the case study on cache replacement, DuelCeptron outperforms SD on recently disclosed Google benchmarks by 2.4% (train set) and 2.9% (test set).

***Index Terms*—cache management, arbitration mechanisms, set dueling, replacement policies, perceptron learning, prefetching**

I. Introduction

Modern CPUs rely heavily on adaptive microarchitectural policies to balance performance efficiency across diverse and rapidly evolving application domains [1]–[3]. Accordingly, microarchitectural mechanisms should operate effectively under highly dynamic program behavior. Because no single policy consistently dominates across the wide diversity of application domains and phase behaviors—each exhibiting different memory access patterns and locality properties—adaptive selection schemes have become essential components of high-end CPU designs, enabling architectures to respond robustly to workload heterogeneity and temporal variability.

Set Dueling (SD) [4][1] has emerged as a widely adopted technique for dynamic policy selection. The appeal of SD lies in its simplicity: it statically assigns a small number of cache sets, typically referred to as *Leader Sets*, to competing policies while the remaining sets, typically referred to as *Follower Sets*, adopt one of the competing policies. The decision is made by using a saturating counter that tracks which Leader Sets performed better. SD is attractive because it is simple, has low hardware cost, and is decoupled from the underlying competing policies. This is evidenced by the broad spectrum of its applications: from general-purpose replacement policies [4], [5], prefetch-aware replacement policies [6], and cache indexing [7] to prefetching [8] and branch prediction [9]. Its influence also extends to real-world CPU designs [1], [3]; reverse engineering of commercial CPUs further reports leader/follower set partitioning that duels two insertion policies in the last-level cache [10], [11]—precisely the structure arbitrated by DRRIP [5], the most widely used SD application.

Despite its great success with numerous applications in both academia and industry, SD has one major limitation: *its decision logic uses a single global counter, which is context-oblivious.* This counter is responsible for dynamically identifying the best-performing policy and applying it to the Follower Sets. Using a single counter implicitly assumes that one of the competing policies will be uniformly superior at any given phase. However, workloads exhibit fine-grained heterogeneity; thus, different policies might be simultaneously useful within the same execution phase. In other words, the problem with using a single global counter is that it reacts only after aggregate hit/miss trends accumulate without capturing contextual correlations, leading to suboptimal policy selection.

Our analysis of emerging Google server workloads [12] shows that augmenting DRRIP [5]—the most common application of SD—with contextual information in its decision logic yields significant performance gains (>2% in geometric mean) over the original context-oblivious SD. These improvements stem from reductions in cache MPKI and access latency. We also demonstrate that these gains persist both with and without established hardware prefetchers [13], [14] and that our findings generalize across standardized benchmarks spanning multiple domains, including SPEC CPU [15], [16], GAP [17], Ligra [18], CVP1 [19], [20], and AI workloads [12].

To address the limitation of SD, we propose *Context-Aware Set Dueling (CASD)*, a general framework that elevates SD to a context-aware, policy-agnostic arbitration mechanism that enables more responsive and accurate runtime adaptation. CASD preserves SD's core training methodology, using Leader Sets to obtain supervised signals about policy effectiveness, but replaces its global monolithic counter with an adaptive and context-aware prediction scheme. Instead of selecting a single policy for all Follower Sets, CASD predicts the most appropriate policy based on execution context (*e.g.*, control flow or data flow derived features). Leader Sets provide training

[1]We use the terms Set Dueling and context-oblivious Set Dueling interchangeably to refer to the original version of Set Dueling.

feedback to the predictor, enabling it to learn correlations between program context and policy effectiveness over time. By introducing contextual learning into SD, CASD provides a simple and general pathway toward more adaptive context-aware microarchitectural policy arbitration; a preliminary version of this idea appeared in [21].

CASD offers several advantages over SD. First, it enables fine-grained adaptation, allowing different contexts within the same execution phase to use different policies concurrently. Second, it improves responsiveness to phase changes, as the predictor can anticipate recurring behavioral patterns rather than relying on slow global counter shifts. More broadly, this work is not about improving a specific application of SD nor proposing a new microarchitectural scheme (*e.g.*, replacement policy); it reframes how microarchitectural arbitration schemes should be designed. Rather than relying on a global decision signal, it demonstrates the benefits of incorporating contextual learning into arbitration, enabling finer-grained and more responsive adaptation across diverse workloads.

This shift toward smarter context-aware microarchitectural mechanisms is increasingly important as architectural performance improvements become harder to achieve due to the slowing of Moore's Law and the growing processor-memory performance gap [22], [23]. As adaptive techniques become pervasive across CPU designs, dynamic arbitration mechanisms will play an increasingly key role in improving performance efficiency across diverse workload domains. In this context, CASD provides a general and extensible framework for designing context-aware arbitration mechanisms. Importantly, these benefits come with low storage overheads since CASD retains SD's low-overhead structure.

To demonstrate the benefits of the CASD framework, we design *DuelCeptron*, a prototype of CASD that uses perceptron learning [24], [25] to correlate contextual information with policy effectiveness and eventually select the best-performing policy for the Follower Sets at any given access. We use three case studies to highlight the effectiveness and versatility of DuelCeptron. First, we apply it on the cache replacement domain and specifically DRRIP [5] since this is the most widely used SD application [3], [10], [11]. The second case study comes from the hardware prefetching domain, where a recent work [8] uses SD to select between two prefetchers. The last study is on prefetch-aware replacement policies. Our evaluation, across a diverse set of applications, shows that DuelCeptron outperforms conventional SD across all considered case studies while incurring low storage overheads.

In summary, this paper makes the following contributions:

- This work reveals that incorporating contextual information in arbitration mechanisms based on Set Dueling (SD) can provide significant performance enhancements and shows a general pathway toward more adaptive context-aware microarchitectural policy arbitration.
- We propose *Context-Aware Set Dueling (CASD)*, a policy-agnostic framework that transforms monolithic SD into a context-aware arbitration mechanism by learning correlations between execution context and policy effectiveness.

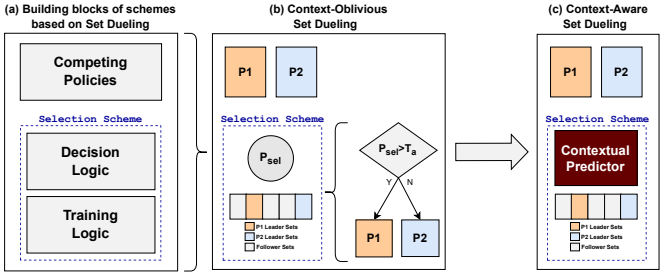

Fig. 1: (a) Building blocks of any microarchitectural mechanism based on SD; (b) Original SD implementation; (c) SD that exploits contextual information in the decision-making.

- We design *DuelCeptron*, a prototype of CASD based on perceptron learning, showing how learning mechanisms can enable accurate and efficient context-aware policy selection.
- We evaluate DuelCeptron across three distinct domains—cache replacement (DRRIP [5]), hardware prefetching [8], and prefetch-aware cache management [6]—and show that it outperforms conventional SD-based arbitration while maintaining low storage overhead. For example, in the DRRIP case study, DuelCeptron improves geometric mean (geomean) performance on recently disclosed Google benchmarks by 2.4% (train set) and 2.9% (test set). We further demonstrate that DuelCeptron provides consistent benefits across a diverse set of workloads (SPEC [15], [16], GAP [17], Ligra [18], CVP1 [19], [20], and AI workloads [12]).

II. Context-Oblivious Set Dueling

Set Dueling (SD) is a generic mechanism for dynamically selecting between competing policies. Figure 1 (a) shows that SD-based schemes fundamentally consist of two building blocks: (1) the competing policies and (2) the selection scheme that dynamically arbitrates between them. The selection scheme comprises (i) the training logic, which gathers runtime information from a subset of resources (*e.g.,* cache sets) and (ii) the decision logic, which interprets this information to determine the superior policy for the current phase and enforce it on the remaining resources (*e.g.,* cache sets).

Figure 1 (b) shows the blocks of the original context-oblivious SD version, which combines two competing policies (P1, P2). Its training logic partitions the corresponding structure (*e.g.*, cache) into three categories: (i) Leader Sets for P1, (ii) Leader Sets for P2, and (iii) Follower Sets. Leader Sets are statically assigned to use one of the competing policies throughout execution. These sets act as controlled experiments and provide feedback under fixed policy behavior. Follower Sets, which constitute the vast majority of the sets, dynamically use one of the competing policies (P1, P2) based on the outcome observed in the Leader Sets. The decision logic that performs this arbitration is typically implemented as a global saturating counter (P_{sel} in Figure 1 (b)). P_{sel} dictates which policy the Follower Sets adopt, enabling the dynamic selection of the superior policy with minimal hardware overhead. In this way, SD separates training (performed by Leader Sets) from deployment (applied by Follower Sets). P_{sel} is compared to an

activation threshold (T_a in Figure 1). Follower Sets use policy P1 when $P_{sel} > T_a$; otherwise, policy P2 is selected. We refer to the original SD as context-oblivious Set Dueling since its decision logic reduces all policy feedback to a global counter, making it incapable of associating policy effectiveness with contextual information.

SD offers several appealing properties: (i) *Low hardware cost*: only a small fraction of sets are designated as leaders, and arbitration requires only a single counter; (ii) *Policy modularity*: SD does not depend on internal details of the competing policies as it treats them as black boxes; (iii) *Adaptation*: SD converges toward the policy that performs better for a given workload. Prior art shows that SD has been successfully adopted across multiple domains due to its simplicity and practicality [5]–[9]; it has been used to arbitrate between replacement policies, prefetch engines, cache indexing schemes, and branch prediction configurations, among others. These properties explain both SD's adoption across domains and its influence on commercial CPU designs.

III. Motivation

This section analyzes the limitations of SD and motivates, both qualitatively and quantitatively, the need to evolve SD into a context-aware arbitration scheme, conceptually presented in Figure 1 (c), which can capture contextual correlations and adapt to fine-grained phase heterogeneity.

A. Limitations of Set Dueling

The simplicity of SD stems from a key design choice: policy selection is global, meaning that all Follower Sets apply the same policy at any given time. While effective for coarse-grained adaptation across workloads or long phases, relying on a single context-oblivious counter limits SD's ability to exploit fine-grained heterogeneity and adapt to changing program behavior. Next, we elaborate on these limitations.

Global Arbitration. The global arbitration of SD implicitly assumes that policy effectiveness is uniform across the entire structure (*e.g.*, cache). In practice, different instruction streams or memory regions within the same execution window may favor different policies—*e.g.*, streaming accesses benefit from aggressive insertion (insert at LRU position or bypass), while temporally reused data benefit from conservative insertion (insert near MRU). Because it collapses all behavior into one counter, SD cannot differentiate these contexts and applies a single policy globally.

Lack of Context Correlation. SD decides solely on aggregate Leader Set outcomes and does not associate policy effectiveness with execution context (*e.g.*, PC or address). Even when a context consistently favors one policy, that preference is reflected only if it dominates the aggregate behavior. Therefore, SD cannot exploit stable, fine-grained regularities (*e.g.*, accesses generated by a static instruction exhibit streaming behavior) and is restricted to coarse-grained trends.

Delayed Adaptation. SD evolves incrementally as its global counter accumulates evidence. When behavior changes, Follower Sets keep applying a potentially suboptimal policy until the counter shifts, degrading performance. This adaptation process is inherently reactive. SD responds only after aggregate behavior has shifted, rather than anticipating recurring patterns. In workloads exhibiting short or interleaved phases, the counter may oscillate or lag behind phase transitions, limiting responsiveness.

B. Quantifying the Potential of Context-Aware Arbitration

The limitations of SD arise from a fundamental structural constraint: SD reduces policy selection to a global decision. Intuitively, addressing this bottleneck requires replacing the scalar global arbitration with a context-sensitive selection mechanism capable of modeling multi-dimensional behavior, *i.e.*, augmenting SD's decision logic with contextual information. The goal of this section is to determine whether doing so can improve the policy selection of SD-based schemes beyond what is achievable with the global counter.

To evaluate our hypothesis, we consider the most widely used scheme based on SD, DRRIP [5], which uses SD to dynamically select between two RRIP-based insertion policies: SRRIP, which inserts at an intermediate re-reference prediction value (RRPV) and is scan-resistant (single-use lines are evicted before the working set), and BRRIP, which inserts most lines at the most distant RRPV and is thrash-resistant (when the working set exceeds the cache capacity, it retains a useful subset). Since scan-dominated and thrash-dominated phases favor opposite policies, DRRIP uses SD to dynamically choose between them. Both SRRIP and BRRIP use the same promotion and eviction policies, differing only in the insertion RRPV of the incoming line; the victim is always the resident line with max-RRPV, independent of the insertion policy.

We compare original DRRIP with a contextual version of DRRIP (DRRIP+Context), which uses the PC as context to drive the policy selection, enabling fine-grained policy arbitration. We use PC as a representative contextual feature since it provides a compact representation of execution context and often correlates with memory access behavior. In practice, DRRIP+Context uses a 1024-entry table (empirically selected) composed of saturating counters, which is indexed with the PC of the memory access. Each counter works like the original SD's global selection counter, but it is local to one PC instead of global. Each table entry says *"Which policy (SRRIP or BRRIP) is better for accesses with this PC?"* Different PCs are indexed in different table entries and thus use different counters, so different contexts can select different policies at the same time. For this motivational study, we assume that each cache line is augmented with the table index used to drive the prediction to ensure correct updates of the table counters. We make this assumption since the PC, which is needed to index the table, is typically not stored in cache lines, and SD updates take place on cache evictions. This assumption is made only for this study; our proposal, presented in Section IV, does not require new cache line metadata. Finally, both DRRIP and DRRIP+Context use the same Leader Sets to enable meaningful comparison; we use 32 Leader Sets per competing policy after performing a sensitivity study.

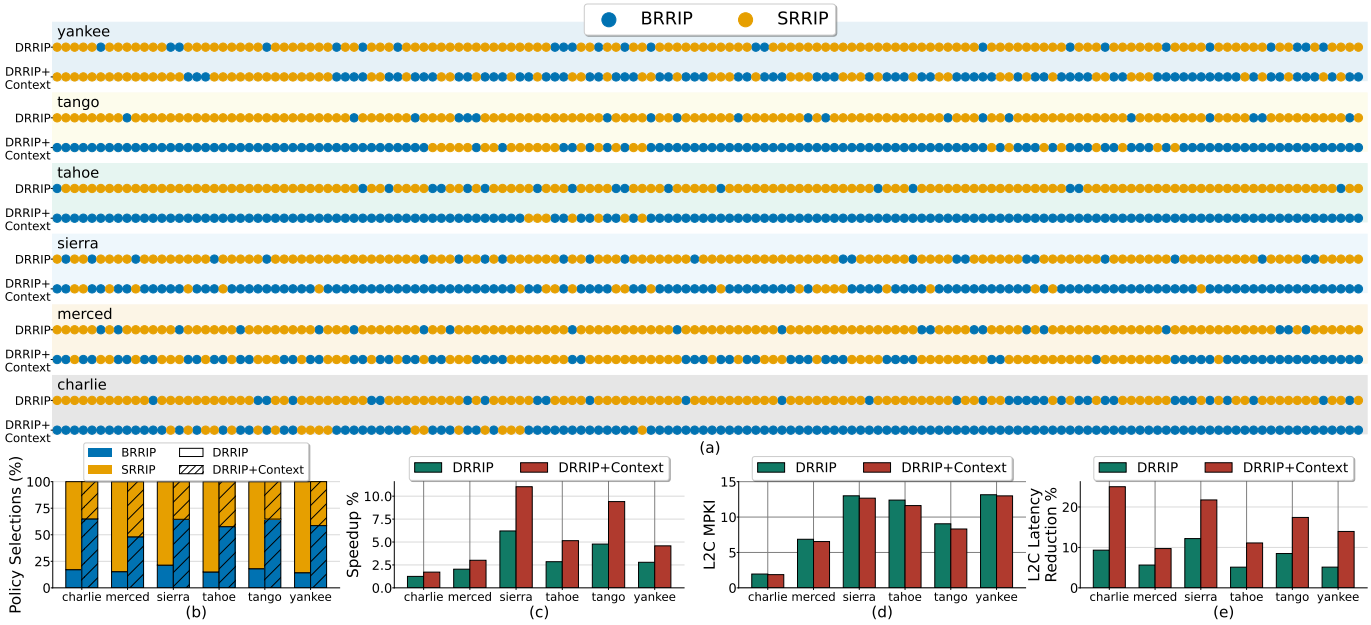

Fig. 2: Comparison between DRRIP and a context-aware version of DRRIP (DRRIP+Context) across a set of Google workloads. (a) Policy selections (BRRIP or SRRIP) made by the arbitration logic for Follower Sets across a sequence of accesses. (b) Fraction of BRRIP and SRRIP selections throughout the execution of each workload. (c) Performance impact (higher is better). (d) Impact on L2C MPKI (lower is better). (e) Impact on access latency reduction (higher is better).

For this study, we use the ChampSim simulator [26], [27] modeling an out-of-order (OoO) CPU with a three-level cache hierarchy. Section V-A presents the details of the simulated system. This motivational study evaluates a set of recent Google workloads exhibiting complex memory access patterns that challenge microarchitectural mechanisms [12]. Our main evaluation, presented in Section V, considers a large set of workloads to highlight the versatility of our proposal.

Figure 2 compares DRRIP with DRRIP+Context when applied to L2C.[2] Focusing on the policy activations (BRRIP or SRRIP) selected by DRRIP and DRRIP+Context, Figure 2 (a) shows that, across a sequence of memory accesses sampled after the warmup period, the two mechanisms exhibit very different activation patterns. For the *tahoe* benchmark, we observe that DRRIP predominantly enables SRRIP, whereas DRRIP+Context mostly selects BRRIP. Another interesting example is *merced*, where DRRIP+Context switches between BRRIP and SRRIP at a much finer granularity; such behavior cannot be achieved by DRRIP due to its reliance on a global counter, which results in slower adaptation to varying execution contexts. The *tahoe* benchmark exhibits mostly thrash behavior over the sampled window, so DRRIP+Context's preference for BRRIP retains a useful working-set subset in the cache while DRRIP, driven by its global counter, does not. The fine-grained alternation of *merced* reflects interleaved scan and thrash phases that favor different policies within the same window of Figure 2 (a); DRRIP+Context captures this behavior using per-context tracking, but DRRIP's global counter cannot.

[2]We focus on L2C rather than LLC because the growing size of L2Cs in modern CPUs [2] makes L2C replacement decisions increasingly impactful.

Overall, Figure 2 (b) shows that DRRIP and DRRIP+Context make substantially different policy decisions: BRRIP (SRRIP) is selected by DRRIP and DRRIP+Context approximately 20% (80%) and 60% (40%) of the time, respectively.

The main finding so far is that DRRIP and DRRIP+Context make different policy decisions. However, this observation alone does not indicate whether one scheme makes better decisions than the other, nor does it reveal their performance impact. To address this, Figure 2 (c) reports the speedups achieved by DRRIP and DRRIP+Context over the baseline. DRRIP+Context consistently outperforms DRRIP in all evaluated Google workloads. This improvement stems from the fact that policy effectiveness is not uniform across a workload: different instruction streams and access patterns favor different policies. By incorporating contextual information into the arbitration process, DRRIP+Context can predict which policy is more suitable for each access and apply different policies to different instruction streams within the same execution phase, rather than enforcing a single global policy.

To better understand the sources of these speedups, Figure 2 (d) and Figure 2 (e) analyze the impact of DRRIP and DRRIP+Context on the L2C MPKI and the average L2C miss latency, respectively. DRRIP+Context achieves both lower L2C MPKI and a greater reduction in the average access latency over the baseline by making more informed insertion decisions—enabling BRRIP or SRRIP depending on which policy is better suited for each access. *The main takeaway of this analysis is that enhancing SD's decision logic with contextual information has the potential to provide significant performance gains due to improved decision-making.*

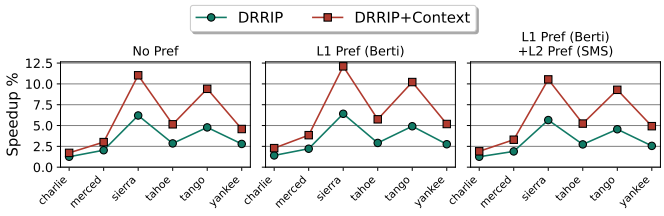

Fig. 3: Performance of DRRIP and DRRIP+Context across different baseline prefetcher configurations.

1) Impact of Hardware Prefetching: The previous section highlights the benefits of making SD context-aware under the assumption that no prefetchers are used. This choice was made to isolate the gains of DRRIP+Context over the original DRRIP. However, modern microarchitectures employ hardware prefetchers to mitigate the Memory Wall bottleneck [22], [23].

In this section, we quantify the benefits of DRRIP+Context over DRRIP in three configurations: (i) no hardware prefetchers, as in the previous section, (ii) the Berti [13] prefetcher operates at L1D, and (iii) Berti is used as the L1D prefetcher and SMS [14] as the L2C prefetcher. Figure 3 presents the speedups achieved by DRRIP and DRRIP+Context over the aforementioned baseline systems, using the same simulation infrastructure and workloads as Section III-B. Despite the integration of prefetching, the performance improvements of DRRIP+Context over DRRIP are consistent and in some cases even enhanced, as in configuration (ii), due to the cache contention introduced by prefetching.

C. Main Takeaways

Our analysis shows that incorporating contextual information into SD's arbitration logic can improve policy selection, with and without prefetching, highlighting the limitations of context-oblivious SD. The contextual feature (PC) used in our motivational study is simple. Nevertheless, it already provides consistent improvements, indicating that incorporating richer contextual information could further enhance the effectiveness of context-aware arbitration. Therefore, the main takeaway of our analysis is that *elevating SD's decision logic into a context-aware prediction framework has the potential to provide significant performance uplifts*.

IV. Context-Aware Set Dueling (CASD)

This work proposes *Context-Aware Set Dueling (CASD)*, a holistic framework that elevates SD to a context-aware, policy-agnostic arbitration mechanism and enables more responsive and accurate runtime adaptation. CASD addresses the limitations of SD identified in Section III-A by transparently incorporating contextual information (*e.g.*, control flow and data flow) into SD's decision logic. By doing so, it can improve the effectiveness of any microarchitectural mechanism that relies on SD. Importantly, CASD remains policy-agnostic, requiring no modifications to the underlying competing policies.

To achieve this, CASD retrofits contextual learning into SD: it preserves SD's original training methodology by using Leader Sets to gather information about policy effectiveness. However, it replaces the traditional global monolithic counter with a context-aware prediction scheme, as illustrated in Figure 1 (c). Instead of selecting a single policy for all Follower Sets, CASD dynamically predicts the most appropriate policy based on the current execution context. Consequently, rather than answering the coarse-grained question *"Which policy performs better overall?"*, CASD addresses the more precise question: *"Which policy performs better under this specific execution context?"* Through contextual learning, CASD provides a simple and general pathway toward more adaptive, context-aware microarchitectural policy arbitration.

A. DuelCeptron: CASD with Perceptrons

This section presents *DuelCeptron*, an implementation prototype of CASD where the decision logic uses a lightweight contextual prediction scheme based on perceptron learning [24], [25]. In other words, DuelCeptron is an instantiation of the contextual predictor of CASD, presented in Figure 1 (c). DuelCeptron removes the structural bottleneck imposed by scalar arbitration of SD as it allows different contexts to favor different policies concurrently, enabling heterogeneous policy deployment across the structure (*e.g.*, cache).

We use perceptron learning for DuelCeptron since perceptron predictors typically have high accuracy due to their ability to correlate predictions with program features while incurring modest storage overheads [28]. Each perceptron predictor of DuelCeptron is realized with a Weight Table (WT) storing perceptron weights associated with one program feature. The perceptron weights are implemented with saturating counters. Note that DuelCeptron employs a hashed perceptron predictor for each selected program feature, leveraging a diverse set of features crafted using both our expertise and prior work [29]–[31]. Section IV-A3 presents the features and the feature selection process. Figure 4 presents the building blocks and the operation in steps (prediction, training) of DuelCeptron.

1) Prediction: At the policy selection event, *i.e.*, when DuelCeptron needs to decide which of the policies to enable, DuelCeptron first extracts the selected program features from the current load request (0 in Figure 4). Then each feature is hashed 1 and used to index the corresponding WT. After indexing, a weight (w_i) is retrieved from the WT 2; the same happens for all considered features. These weights represent the learned preference between the competing policies for that context. Next, all weights pass through the Weight Optimizer, which is a module responsible for adjusting the weights of all features by taking into account the system state and phase behavior 3. This module changes the importance of a feature (shift operation, no floating-point) by examining the code cache accesses. To do so, it uses the metric Code Accesses per Kilo Instructions (CAKI) 3.1. In practice, if CAKI exceeds a high threshold T_{high}, the Weight Optimizer shifts right the weights of features using only data-flow information, giving higher priority to features using control-flow. Similarly, if CAKI falls below a low threshold T_{low}, the Weight Optimizer shifts right the weights of features using only control-flow information, giving higher priority to features using data-flow.

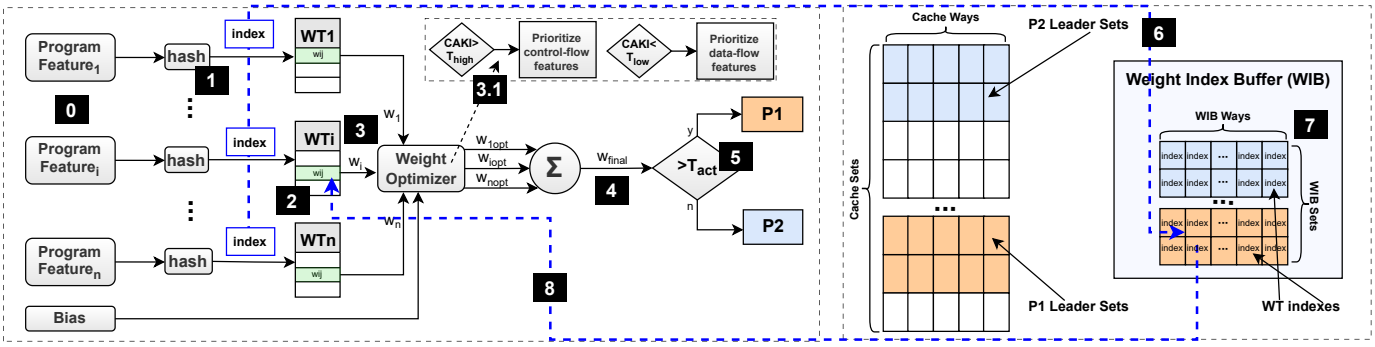

Fig. 4: DuelCeptron design and operation in steps.

After the weight optimization step, DuelCeptron sums the optimized weights of all considered features and generates the final weight (w_{final}) 4. The final step includes comparing w_{final} with an activation threshold T_{act} 5. If w_{final} is greater than T_{act}, policy P1 is selected; otherwise, policy P2 is used for the current access. Through this process, DuelCeptron dynamically selects different policies for the Follower Sets based on contextual evidence rather than global aggregation.

2) Training: DuelCeptron must ensure correct updates of the selected feature weights stored in the WTs. To achieve this, it introduces a new structure called the Weight Index Buffer (WIB), which stores the hash indexes of the selected features used during prediction, allowing DuelCeptron to correctly update the corresponding weights when training occurs (6 in Figure 4). Note that DuelCeptron (and any CASD prototype) uses the same training signals as the original SD, *i.e.*, the events that update SD's selection counter. For example, in the DRRIP case study (Section V), the weights of DuelCeptron are updated upon Leader Set evictions using the WIB (operation described below). Optimizing the training signals could improve the performance of DuelCeptron, but we aim for a fair comparison with the original SD; therefore, both SD and DuelCeptron use the same training signals.

WIB only stores the hash indexes of the Leader Sets for all competing policies, since these sets form the training environment for DuelCeptron, as shown in 7. Each Leader Set resident line therefore maps to exactly one WIB entry; thus there is no associative tag match and no aliasing. In a straightforward implementation, WIB has as many sets as the Leader Sets, with associativity matching that of the underlying cache. Section V-D shows that both the number of Leader Sets and the WIB associativity can be reduced, *i.e.*, DuelCeptron can be trained on a subset of cache ways and Leader Sets while preserving its performance gains and lowering storage overhead. For now, we assume that the WIB matches the associativity of the underlying cache, unless stated otherwise.

WIB is needed when DuelCeptron employs control-flow–based features (*e.g.*, the PC). In this case, feature values used at prediction time are not available during training because they are not typically stored per cache line. By recording the hashed feature indexes in the WIB, DuelCeptron retains the information required to correctly update the corresponding weights during training. If DuelCeptron is restricted to features whose required metadata for training is already maintained per cache line, the WIB can be eliminated. For completeness, we consider WIB in our proposal.

When a training event occurs (*e.g.*, a cache eviction from the Leader Sets for DRRIP), DuelCeptron adjusts the confidence of the competing policies for that context. To do so, it retrieves the hashed indexes of the corresponding way stored in the WIB to update the perceptron weights 8. In practice, the counters associated with these weights are adjusted—incremented or decremented—toward the policy assigned to the corresponding Leader Set. For example, in DRRIP [5], consider an eviction from the Leader Set assigned to policy P1 (negative training for P1). In this scenario, DuelCeptron uses the hashed indexes stored in the WIB to decrease the corresponding weights, indicating less confidence for P1 under this context. As a result, when the same context appears again, DuelCeptron may shift toward favoring policy P2 instead (5 in Figure 4). Finally, these updates are bounded by the saturation limits of the counter width to ensure stability.

3) Feature Selection: DuelCeptron selects from a wide range of 30 program features, crafted using our expertise and prior work [29]–[31], that can be used to design a context-aware arbitration mechanism. Table I presents the subset of program features, identified by offline feature exploration, that provide the highest performance gains in isolation among the evaluated case studies of DuelCeptron. We observe that most program features use the address, PC, and combinations of them. Note that DuelCeptron does not contain features specialized to the application domain (*e.g.*, cache replacement policy and DRRIP), but rather features that are agnostic to where DuelCeptron is applied. Crafting specialized features that exploit metadata from specific SD application domains (replacement policies [4], [5], indexing schemes [7], prefetching [8], branch prediction [9]) can further improve the effectiveness of DuelCeptron. Our work does not focus on a specific SD application but rather provides a holistic framework for elevating SD into a context-aware arbitration mechanism.

4) Feature Combination: The feature selection of Section IV-A3 is performed offline and uses performance as the optimization metric as it is a global metric applicable across all case studies and is commonly used in prior work [29], [31].

Feature	Description
PC/PC≫12	PC/shifted PC of the memory access
$PC_{i-2}⊕PC_{i-1}⊕PC_i$	Last three PCs
PA/PA≫12	Physical address/page of the access
$PA_{i-2}⊕PA_{i-1}⊕PA_i$	Last three physical addresses
$(PA_{i-2}≫12)⊕(PA_{i-1}≫12)⊕(PA_i≫12)$	Last three pages touched
PC⊕CacheLineOffset	PC XORed with cache line offset
PC⊕(PA≫12)	PC XORed with physical page
PC⊕PA	PC XORed with physical address

TABLE I: Best-performing features among the 30 examined.

First, we quantify the speedup of each program feature in isolation by evaluating single-feature DuelCeptron instantiations (Table I) across the training workloads. The features are then sorted by increasing geomean speedup. In the final step, we combine features to further improve performance. We start with a set containing only the best-performing feature and iteratively evaluate the addition of the remaining features. If adding a feature improves the geomean speedup by more than 0.3% over the best configuration so far, it is included in the set of selected features. This process continues until all features have been examined. Because this greedy search evaluates only a subset of the joint feature space, scaling such explorations is a design-methodology problem, for which agentic LLM workflows are a promising direction [32].

5) Storage Overhead: The storage overhead of DuelCeptron depends on the number of competing policies (M), the number of selected features (N), the size of WTs (WT_i^{size}), and the number of bits used for perceptron weights ($bits^{WT_i}$). It also depends on the number of Leader Sets per competing policy (L_{sets}) and the number of WIB ways (WIB_{ways}), since these parameters impact the WIB size (see 7 in Figure 4). The total storage overhead of DuelCeptron is given by Equation 1. The first term accounts for the storage of WTs and the second term represents the bits needed for storing the perceptron indices in WIB to ensure correct weight updates.

$$\sum_{i=1}^{N} WT_i^{size} \cdot bits^{WT_i} + M \cdot L_{sets} \cdot WIB_{ways} \cdot \sum_{i=1}^{N} \log_2(WT_i^{size}) \quad (1)$$

B. Discussion

1) Other Models: Our prototype employs perceptron learning to map program context to policy selection, offering a practical balance between hardware cost, generality, and feature combination via linear weighting. More expressive learning mechanisms [33]–[35] could capture richer context–policy relationships, but at higher complexity.

2) Number of Competing Policies: Both SD and CASD assume two competing policies. Extending to M competing policies is feasible but requires additional Leader Sets and a multi-class selection mechanism: an M-way maximum over the per-policy weight sums, which is combinational and off the critical path (Section V-D1, as for the two-policy case). The per-update training cost stays $O(features)$, independent of M, since each event updates only the weights of the evicted line's features. However, more Leader Sets reduce the number of Follower Sets and enlarge the decision space, potentially slowing convergence. Prior work shows that arbitrating three policies with a counter-driven decision tree provides benefits [36]; whether such gains persist once arbitration is context-aware remains open.

3) Concurrent Policy Coexistence: CASD inherits SD's precondition that the competing policies coexist concurrently, so CASD applies wherever SD applies, without modifying the underlying policies. One consideration is switching granularity: CASD switches per access, so for the class of high-inertia policies such as cache indexing schemes [37], where reconfiguration can require invalidating or relocating lines, the dominant cost shifts from training to per-switch reconfiguration. The large majority of SD applications do not face this, as their competing policies differ only in lightweight per-request decisions that coexist at no switching cost. For the high-inertia case, CASD admits a near-free extension: a configurable right-shift on the feature-weight sum (reusing the Weight Optimizer's shift primitive) attenuates the per-context contribution so the SD-like bias dominates, bounding switching frequency toward conventional SD.

Component	Configuration
CPU Core	4GHz, 352-entry ROB, hashed perceptron branch predictor [28]
L1 ITLB/DTLB	64/128-entry, 4/4-way, 1/1-cycle, 8-entry MSHR, LRU
L2 TLB	2048-entry, 16-way, 8-cycle, 16-entry MSHR, LRU
L1I	48KB, 12-way, 5-cycle, 8-entry MSHR, LRU
L1D	48KB, 12-way, 5-cycle, 16-entry MSHR, LRU, Berti [13]
L2C	1.25MB, 20-way, 10-cycle, 32-entry MSHR, LRU, SMS [14]
DuelCeptron	PC≫12: 256-entry WT, 9b/entry, $(PA_{i-2}≫12)⊕(PA_{i-1}≫12)⊕(PA_i≫12)$: 256-entry WT, 7b/entry, PC⊕CacheLineOffset: 512-entry WT, 7b/entry, L_{sets}=32 (per policy), $T_{act} = 240$, $T_{high} = 30$, $T_{low} = 0.2$
LLC (per core)	2MB, 16-way, 20-cycle, 64-entry MSHR, LRU
DRAM	8GB, 3200MT/s

TABLE II: Simulated system and DuelCeptron configuration.

V. Case Study 1: Cache Replacement Policy

To evaluate the benefits of our proposal, CASD, and its prototype, DuelCeptron, we first focus on the most common application domain of SD: cache replacement policies, and specifically DRRIP [5], similar to Section III-B. This case study constitutes our main evaluation. Section VI and Section VII demonstrate that DuelCeptron also improves performance in different SD application domains.

A. Methodology

Simulation Infrastructure. Our evaluation uses the ChampSim simulator [26], [27], modeling an OoO CPU with a three-level cache hierarchy, a decoupled front-end [38], and an MMU model with hardware page walking [39], [40] and MMU caches [41]. Table II summarizes the simulated system.

Workloads. We evaluate a diverse set of workloads from different benchmark suites and application domains. We use (i) emerging server workloads provided by Google [12], (ii)

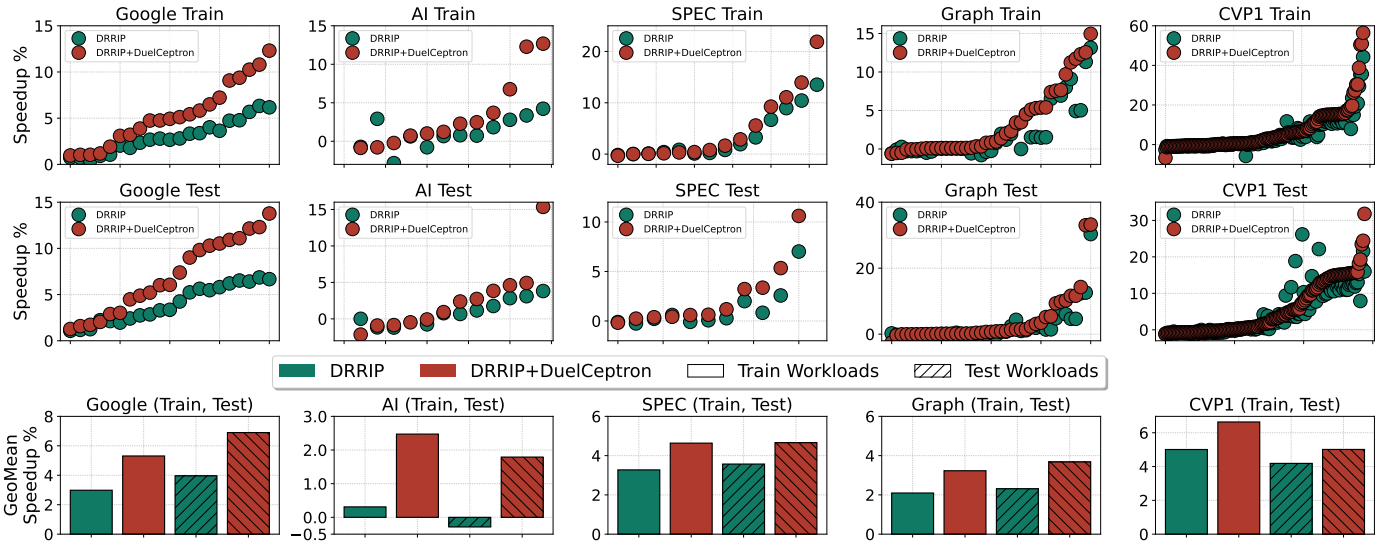

Fig. 5: Performance of DRRIP and DRRIP+DuelCeptron across the train set and test set of workloads.

AI workloads (*e.g.*, llama2, vit, whisper) [12], (iii) general-purpose workloads from SPEC CPU 2006 [15] and SPEC CPU 2017 [16]—we refer to them as SPEC, (iv) graph workloads included in the GAP [17] and Ligra [18] benchmark suites—we refer to them as Graph, (v) industrial integer, floating point, and server workloads provided by Qualcomm for CVP1 [19], [20], and (vi) established server workloads (NodeApp, TPCC, Wikipedia, Kafka, Spring, Tomcat, Chirper, HTTP) [42], [43]—we refer to them as SRV. All traces were obtained using the SimPoint methodology [44]. SimPoints with a weight smaller than 0.05 are discarded and we report weighted geomean speedups [8], [35], [45], [46]. SPEC, GAP, and Ligra workloads execute 250M instructions for warmup followed by 250M instructions for simulation. CVP1 workloads use 50M warmup instructions and 100M simulation instructions [47]–[49]. Google and AI workloads use 50M warmup instructions followed by 200M simulation instructions [12].

Our evaluation focuses on workloads with LLC MPKI>1, since replacement policies have a negligible impact on compute-bound workloads. In total, we consider 476 workloads, which we divide into three categories. The train set guides the design of DuelCeptron, while the test set is held out from development; we form both by splitting the Google, AI, SPEC, Graph, and CVP1 suites evenly, assigning half of each suite to training and the remainder to test. The third category comprises the SRV workloads that appear in neither set and thus represent an unseen workload class.

1) Multi-core Methodology: Our multi-core evaluation considers 400 randomly generated mixes from our single-core workload set. Our evaluation reports the weighted speedup over the baseline [8], [29]–[31], [50]. First, we compute the IPC in the multi-core context and the IPC in isolation on a system with the multi-core configuration for each application running on a core. Next, we compute the weighted IPC as the sum of (IPC_{mc}/IPC_{iso}) for all workloads in the mix. Finally, we normalize this sum with the weighted IPC of the baseline.

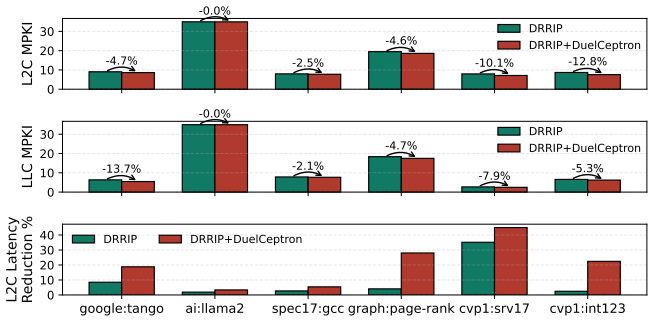

Fig. 6: Impact on cache MPKIs and access latency.

For each mix, when a core finishes its instructions, it gets replayed until all the cores finish their respective instructions, similar to prior work [13], [31], [45], [51], [52].

2) SMT Methodology: To evaluate DuelCeptron under thread colocation, we construct 50 randomly selected workload pairs from the pool of single-core workloads to capture diverse interference scenarios. We use the same warmup and simulation instructions as in the single-core experiments.

B. DuelCeptron Configuration

Following the feature selection methodology described in Section IV-A3, we configure DuelCeptron with three features from Table I that capture complementary control-flow and data-flow information that correlate with policy effectiveness.

1) **`PC≫12`** is computed by shifting the program counter right by 12 bits. This provides control-flow information and allows DuelCeptron to learn whether accesses originating from specific code regions favor a particular policy (SRRIP or BRRIP in this case study).

2) **`Last 3 Pages`** captures data-flow behavior and is computed by storing the last two physical pages accessed (after shifting the addresses right by 12 bits) and XORing them with the currently accessed physical page. This feature enables

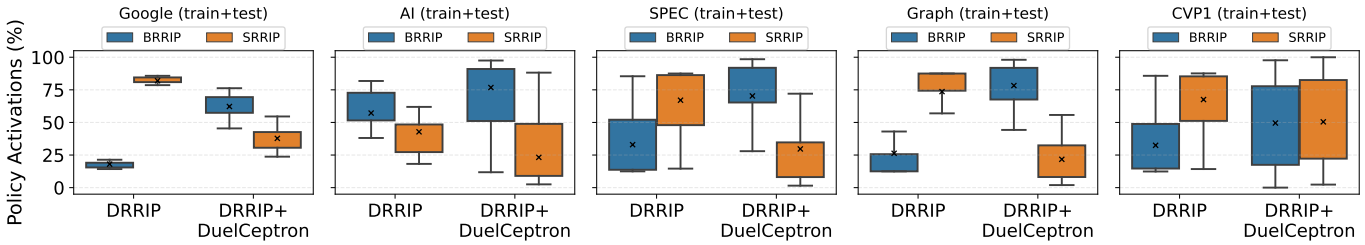

Fig. 7: Percentage of BRRIP and SRRIP activations by DRRIP and DRRIP+DuelCeptron across all workloads.

DuelCeptron to capture correlations between sequences of accessed pages and the effectiveness of the competing policies.

3) **PC⊕CacheLineOffset** combines control-flow and data-flow information by XORing PC with the cache line offset. This allows DuelCeptron to capture finer-grained correlations where both the PC and the line offset influence which policy performs best for a given access.

C. Performance Evaluation

Figure 5 compares the performance of DRRIP and DuelCeptron (DRRIP+DuelCeptron),[3] which replaces the monolithic arbitration logic of SD with a contextual perceptron-based prediction scheme over a baseline that uses LRU as the L2C replacement policy (Table II) across the Google, AI, SPEC, Graph, and CVP1 workloads, presented in Section V-A. The first and second rows of Figure 5 present results for the train and test workloads, respectively, while the third row shows the geomean speedups for both workload sets. The main observation is that DRRIP+DuelCeptron consistently improves performance over DRRIP across all examined workload domains (for both train and test workloads).

Overall, DRRIP+DuelCeptron outperforms DRRIP for the Google train/test, AI train/test, SPEC train/test, Graph train/test, and CVP1 train/test by 2.4%/2.9%, 2.5%/2.1%, 1.4%/1.1%, 1.2%/1.4%, and 1.6%/1.0%, respectively. The improvements are comparable between the train workloads used during development and the test workloads, indicating that DuelCeptron generalizes well to unseen applications and does not overfit to the workloads used during tuning. Importantly, DuelCeptron uses the same configuration, presented in Section V-B and Table II, for all workloads and is not tuned per benchmark suite. Although domain-specific tuning (*e.g.*, WT sizes, bit width, thresholds) could yield higher gains, the goal of this work is to demonstrate that augmenting SD with a contextual arbitration scheme can provide consistent performance improvements in diverse applications rather than optimizing the design for a specific benchmark suite.

1) Impact on MPKI and Access Latency: The performance gains achieved by DuelCeptron are driven by reductions in cache MPKI and access latency. Figure 6 illustrates this by comparing the impact of DRRIP and DRRIP+DuelCeptron on L2C MPKI, LLC MPKI, and L2C access latency relative to the baseline. This study focuses on representative workloads from each benchmark suite, rather than all 476 evaluated workloads.

[3]DRRIP and DRRIP+DuelCeptron use the same 32 Leader Sets per competing policy (empirically selected after tuning).

Figure 6 shows that DRRIP+DuelCeptron significantly reduces L2C MPKI for most workloads (*e.g.*, tango and srv17). This trend extends to the LLC, where MPKI reductions are also consistently observed. This behavior is expected: improving L2C replacement decisions reduces the number of accesses that miss in L2C and propagate to the LLC, thereby lowering LLC pressure and miss rates. Similarly, L2C access latency is reduced more under DRRIP+DuelCeptron than under DRRIP, as more accesses are served from the cache hierarchy. These improvements stem from more accurate policy selection decisions enabled by correlating execution context with the effectiveness of the competing insertion policies.

To quantify this behavior across all workloads (both train and test sets), we measured that DRRIP+DuelCeptron reduces L2C MPKI by more than 1% for 69.7% of the workloads. Among these, 98.2% also exhibit reductions in LLC MPKI, and 71.0% achieve reductions in average L2C access latency. For the remaining 30.3% of workloads, DRRIP+DuelCeptron still reduces LLC MPKI in 83.6% of cases and improves L2C access latency in 95.8% of cases. This indicates that even when L2C MPKI is not significantly reduced, DuelCeptron improves cache effectiveness by retaining performance-critical cache lines in L2C, leading to lower access latency.

These MPKI and latency reductions translate directly into the observed performance gains by decreasing long-latency memory accesses. A key advantage of CASD and DuelCeptron over original SD is its ability to select different policies for different execution contexts within the same program phase. While DRRIP relies on a single global counter and therefore applies the same policy to all Follower Sets, DuelCeptron uses contextual information to select the insertion policy on a per-access basis. Our analysis (Section III-B) showed that different contexts within the same workload often favor different policies. By capturing these correlations, DuelCeptron allows SRRIP and BRRIP to be applied simultaneously to the contexts where they are most effective, leading to improved cache utilization. Because DuelCeptron selects only the insertion policy, each per-access selection sets the incoming line's own insertion priority, *i.e.*, its RRPV, and never selects the victim; SRRIP- and BRRIP-inserted lines coexist within a set, where standard max-RRPV eviction operates over their combined state, as explained in Section III-B.

Figure 7 supports our analysis finding by showing the activations of BRRIP and SRRIP across the considered suites. While DRRIP tends to bias toward a single policy, DuelCeptron distributes activations across both policies within the same

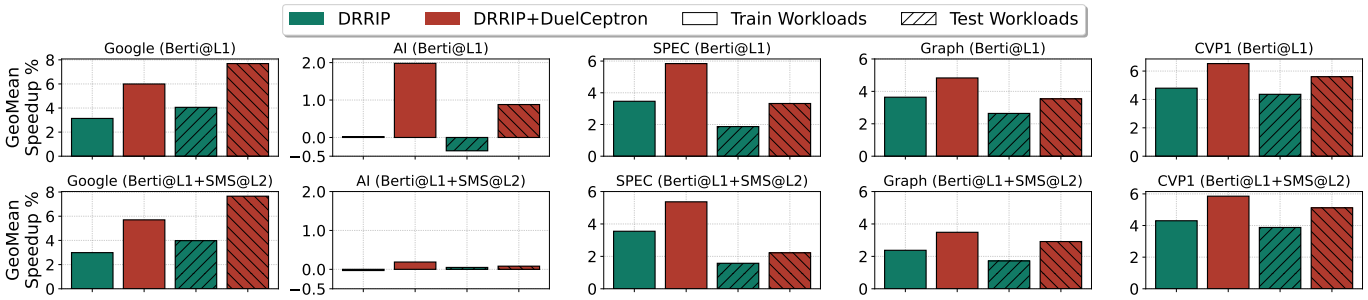

Fig. 8: Performance of DRRIP and DRRIP+DuelCeptron across different prefetcher configurations (Berti at L1D in row 1 and Berti at L1D and SMS at L2C in row 2). Reported speedups are for both train and test workloads.

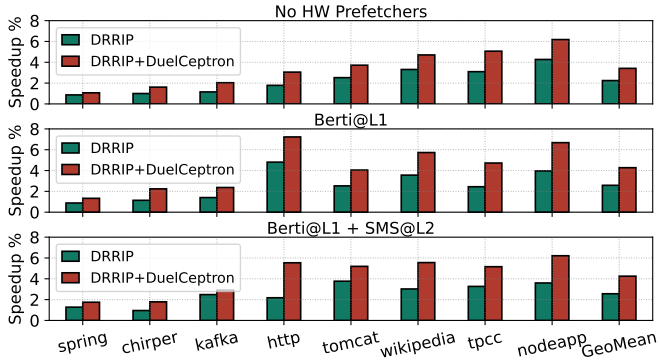

Fig. 9: Performance of DRRIP and DRRIP+DuelCeptron when applied to a completely unseen class of workloads.

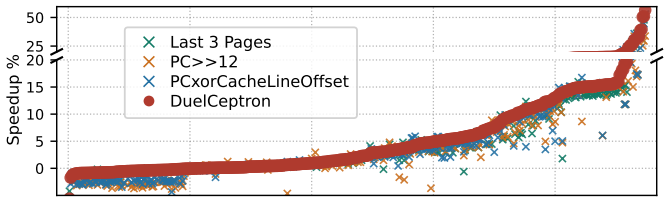

Fig. 10: Performance comparison between DuelCeptron and its constituent features across all considered workloads.

workload domain. This indicates that DuelCeptron captures fine-grained behavioral differences across execution contexts and dynamically assigns the most suitable policy per access. The wider spread and overlap of BRRIP and SRRIP activations across all suites further confirm that different contexts within the same workload benefit from different insertion strategies, validating the need for context-aware arbitration.

2) Impact of Hardware Prefetching: So far, we assume no hardware cache prefetchers to isolate the comparison between DRRIP and DRRIP+DuelCeptron. Next, we evaluate the interaction between DuelCeptron and hardware prefetching. Figure 8 reports the geomean speedups of DRRIP and DRRIP+DuelCeptron for two baseline configurations with prefetchers: 1) Berti [13] operates at L1D and 2) Berti operates at L1D and SMS [14] at L2C. We observe that, across both configurations with prefetching, DRRIP+DuelCeptron consistently outperforms DRRIP for all workload classes (both train and test sets). The magnitude of the improvements remains similar to the no-prefetching baseline (Figure 5), suggesting that the benefits of DuelCeptron are orthogonal to the presence of hardware prefetchers. For example, DRRIP+DuelCeptron outperforms DRRIP for the Google train/test workloads by 2.4%/2.9%, 2.9%/3.6%, and 2.7%/3.6% when the baseline has no, Berti, and Berti+SMS prefetchers, respectively. An exception occurs for the AI workloads when SMS is used at L2C. In this case, both DRRIP and DRRIP+DuelCeptron provide limited gains because these AI applications mainly exhibit spatial patterns that are effectively captured by SMS, reducing the sensitivity of performance to the replacement policy. Overall, this study provides additional confidence that no matter the configuration, DuelCeptron provides significant gains, highlighting the benefits of transforming SD into a context-aware arbitration framework.

3) Unseen Workload Class: To further evaluate the ability of our proposal to generalize beyond the workloads used during development, we evaluate DuelCeptron on the SRV workloads described in Section V-A. These workloads are not included in either the train or test sets and thus represent an unseen workload class. Figure 9 shows that DRRIP+DuelCeptron consistently outperforms DRRIP across all examined baseline configurations, including no prefetching, Berti at L1D, and Berti at L1D combined with SMS at L2C. The improvements are consistent with the trends observed on the train and test workload sets, indicating that DuelCeptron does not overfit to the workloads used during design and that its contextual arbitration mechanism generalizes to unseen workloads.

4) Comparison with Individual Features: This section compares DuelCeptron with its constituent features (PC$\gg$12, Last 3 Pages, PC$\oplus$CacheLineOffset). To do so, we implement three versions of DuelCeptron that use only one of DuelCeptron's features. Figure 10 presents the comparison. DuelCeptron outperforms every single-feature version on the vast majority of workloads, because it combines the benefits of the individual features, with each feature helping the workloads it correlates with best. Crucially, DuelCeptron is not biased toward any single feature: specific features dominate the decisions only on the workloads, or workload phases, whose behavior they capture well, while others dominate elsewhere. For example, feature PC$\gg$12 is very important for many Google workloads while being neutral for many SPEC workloads. Designs with larger storage budgets could incorporate additional features

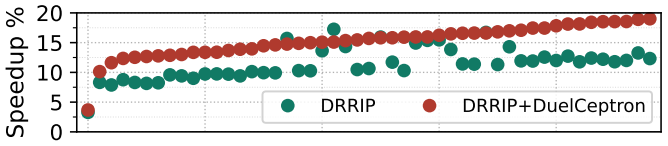

Fig. 11: Performance of DRRIP and DRRIP+DuelCeptron across 50 SMT (2 hardware threads) workloads.

from Table I that provide gains for certain workloads but have limited impact on overall geomean performance.

5) SMT Evaluation: DuelCeptron is thread-agnostic since WTs are indexed by program features (*e.g.*, PC), not by thread, so SMT interleaving is simply a mix of feature values, and each training update depends only on the evicted line, not on state accumulated across accesses. Interleaved updates therefore remain correct, and the saturating-counter hysteresis absorbs the additional variance from interleaving.

Figure 11 compares DuelCeptron and DRRIP across 50 SMT workloads (Section V-A2), with two hardware threads sharing the cache hierarchy. DuelCeptron outperforms DRRIP on most workloads (4.2% on average). The results follow the same trends as the single-thread evaluation but with larger absolute speedups, because the two hardware threads increase cache contention, thus increasing the potential gains of a better replacement policy. A risk under thread colocation is destructive aliasing between threads that collide on the same feature hash with opposing policy preferences; where significant, this can be addressed by folding a thread ID into the feature hash, leaving training and arbitration unchanged.

D. Storage and Latency Overheads

Results presented so far use the DuelCeptron configuration shown in Table II. The storage overhead of this configuration is 4.8KB (0.9KB for the WTs and 3.9KB for the WIB), computed using Equation 1, corresponding to 0.4% of the simulated L2C. The features PC$\gg$12, Last 3 Pages, and PC$\oplus$CacheLineOffset use WTs with 256, 256, and 512 entries, respectively, empirically selected after tuning. DuelCeptron uses 64 Leader Sets (32 for SRRIP and 32 for BRRIP), consistent with DRRIP and identified as the best-performing configuration. The overall storage overhead of DuelCeptron (Equation 1) is primarily determined by two parameters: the number of Leader Sets and the number of WIB ways. While we use 64 Leader Sets—since this configuration is optimal for DRRIP—DuelCeptron does not inherently depend on this choice. To quantify the trade-offs between storage and performance, we perform two sensitivity studies. First, we vary the number of Leader Sets while keeping the number of WIB ways constant. Second, we vary the number of WIB ways while keeping the number of Leader Sets constant; this study effectively performs way sampling within the Leader Sets (7 in Figure 4).

Figure 12 (a) shows that 64 total Leader Sets provide the best performance, while reducing them to 32 results in a modest geomean performance loss. Further reductions in the number of total Leader Sets, however, significantly degrade performance. Figure 12 (b) shows that sampling 15 or 10 ways (out of 20) achieves nearly identical speedups while substantially reducing storage overhead. Finally, Figure 12 (c) combines the insights of the previous studies and demonstrates that the configuration of DuelCeptron with 32 total Leader Sets and 10 WIB ways achieves almost the same speedups as the configuration with 64 total Leader Sets and 20 WIB ways, while reducing storage by 2.5× (from 4.8KB to 1.9KB).

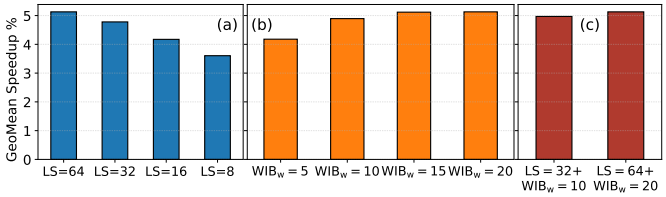

Fig. 12: (a) Effect of total Leader Set (LS) count on DuelCeptron performance; (b) Effect of WIB ways (WIB$_w$) on DuelCeptron performance; (c) Comparing storage-efficient DuelCeptron (left) with original DuelCeptron (right).

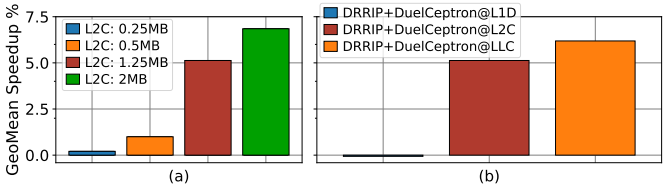

Fig. 13: Impact of (a) L2C size and (b) the cache level at which DuelCeptron is applied on DuelCeptron's speedup.

The results in Figure 12 indicate that DuelCeptron does not require a large number of Leader Sets or WIB ways to learn correlations between access patterns and policy effectiveness. Therefore, we advocate for a storage-efficient configuration with 32 total Leader Sets and 50% of the baseline cache ways used as WIB ways, as a practical design point.

1) Prediction and Training Latencies: DuelCeptron predicts which of the two policies (SRRIP, BRRIP) to enable at line insertion, not on the cache access path. Insertion occurs only after an L2C miss has been serviced (tens of cycles), and the features needed for prediction (*e.g.*, PC, physical address) are available the moment the demand miss is issued. The prediction is therefore computed in parallel with miss servicing and is ready well before the fill data returns. We verified this by sweeping DuelCeptron's prediction latency: results yielded identical performance, because the prediction is fully overlapped with miss servicing and off the critical path. Thus, our proposal does not stall the fill. Training is equally undemanding: updates occur only on Leader-Set evictions, off the access and fill paths, and each adjusts a few saturating counters by ± 1; a recurring context reaches a confident weight within tens of such events, while the 7–9-bit counters provide hysteresis that prevents oscillation under transient noise.

E. Sensitivity Studies

Figure 13 (a) examines the impact of the L2C size on DuelCeptron's performance across all workloads (train and test

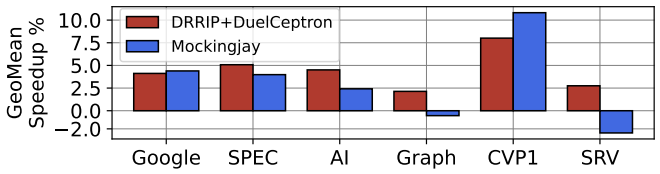

Fig. 14: Comparison with state-of-the-art.

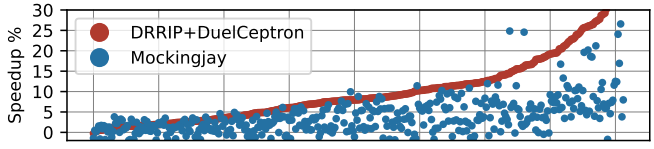

Fig. 15: Multi-core evaluation across 400 randomly generated 4-core mixes. Each marker represents a mix.

sets). The gains of DRRIP+DuelCeptron increase with larger L2C sizes. This trend is expected, as replacement policies have greater impact when more capacity is available to manage.

Figure 13 (b) evaluates the effect of applying DRRIP+DuelCeptron at different cache levels (L1D, L2C, LLC) across all considered workloads (train and test sets). Applying the policy at L1D yields no performance improvement due to the limited size of L1D—consistent with the observations from varying L2C size in Figure 13 (a). Finally, applying DRRIP+DuelCeptron at the LLC results in slightly higher performance gains than applying it at L2C.

F. Comparison with State-of-the-Art

Although the primary goal of this work is not to propose a new replacement policy, we compare DRRIP+DuelCeptron with the state-of-the-art, Mockingjay [53], in Figure 14 to assess the impact of incorporating contextual arbitration against the state-of-the-art. We apply DRRIP+DuelCeptron at the LLC, similar to Section V-E, matching Mockingjay's design point. Across all train and test sets and the unseen SRV workloads, DRRIP+DuelCeptron matches or outperforms Mockingjay on all workload classes, with CVP1 being the only exception. This is notable given that DuelCeptron improves only the arbitration of a simple, widely-used policy without any modification to the underlying policy logic.

Mockingjay predicts per-line reuse distance while DuelCeptron arbitrates the insertion policy from execution context. DuelCeptron is more effective where context cleanly separates cache-friendly from scan/thrash behavior, and less effective where fine-grained per-line reuse-distance estimation is the dominant signal. The AI workloads exhibit context-predictable access patterns that favor contextual arbitration, which explains their larger gains. Mockingjay degrades performance for the SRV workloads, while DRRIP+DuelCeptron provides speedups. This happens because the SRV workloads have large code footprints with irregular patterns that are poorly served by per-line reuse-distance estimation, whereas context-based arbitration still identifies useful insertion behavior.

Mockingjay and DRRIP+DuelCeptron also differ in storage overhead. Mockingjay requires 31.9KB while DRRIP+DuelCeptron needs 4.1KB (1.7KB in the storage-efficient version of Section V-D)—∼7.9× (∼18.6×) less—computed with Equation 1 for the LLC configuration of Table II. DuelCeptron's cost is set by the number of Leader Sets and WIB ways rather than by cache capacity, so it barely changes across cache levels (4.8KB at L2C, 4.1KB at LLC). This comparison highlights that improving the arbitration mechanism alone can close the gap with the state-of-the-art, and points to a promising direction for replacement policy design.

1) Multi-core Evaluation: Shared LLC contention is amplified in multi-core contexts; thus we extend the comparison with Mockingjay to a 4-core setting with Berti as L1D prefetcher and SMS as L2C prefetcher, similar to Section V-C2. Figure 15 compares DRRIP+DuelCeptron with Mockingjay across 400 randomly generated 4-core mixes (Section V-A1). DRRIP+DuelCeptron outperforms Mockingjay on 370 of the 400 mixes, improving average performance by 9.6% versus Mockingjay's 3.8%. These results indicate that contextual arbitration scales with multi-core pressure better than per-line reuse-distance prediction.

VI. Case Study 2: Hardware Prefetching

To further highlight CASD's effectiveness in different SD domains, we consider as a second case study a work on prefetching [8]. This work propagates the page size information to the lower-level prefetchers, enabling safe prefetching beyond 4KB physical pages when the accessed blocks reside on large pages. A prefetcher that exploits this module is referred to as Page Size Aware Prefetcher (Pref-PSA). The authors show that incorporating large pages in the design and the operation of a prefetcher may improve or degrade performance, depending on the application properties. To exploit the performance gains offered by incorporating the notion of large pages in the design of a prefetcher without harming the performance of workloads that do not benefit from it, they propose a composite scheme that uses SD (as presented in Section II) to dynamically select the best-performing prefetcher between Pref-PSA, which assumes 4KB pages to drive prefetching, and a Pref-PSA utilizing large pages of 2MB, referred to as Pref-PSA-2MB.

This section examines whether replacing SD's global arbitration scheme from this work [8] with DuelCeptron can provide more benefits due to context-aware arbitration between Pref-PSA and Pref-PSA-2MB. To ensure a directly comparable evaluation, we use the same simulation infrastructure, baseline, and benchmarks used in the original work.

DuelCeptron Configuration. For configuring DuelCeptron, we follow the feature selection process described in Section IV-A3 and Section IV-A4, similar to the previous case study. Our experiments indicate that the best-performing features for this case study are the `PC` and the `Physical Address (PA)` from Table I. Both features use 512-entry WTs and a 5-bit weight per WT entry. The storage overhead of this configuration is 1.75KB.

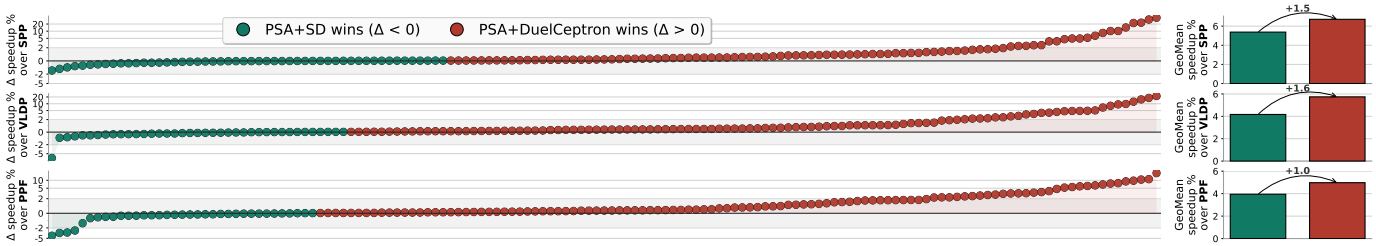

Fig. 16: Performance comparison between PSA Prefetching with Set Dueling (PSA+SD) [8] and PSA Prefetching with DuelCeptron (PSA+DuelCeptron). The speedups are computed over the original version of the underlying prefetcher.

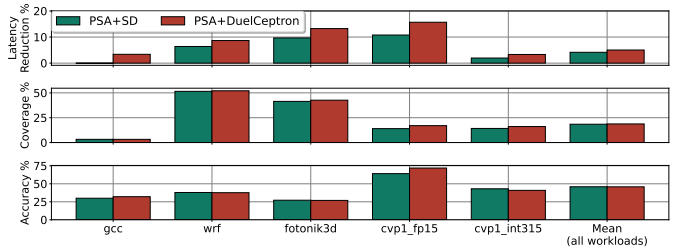

Fig. 17: Impact of PSA+SD and PSA+DuelCeptron versions of SPP on access latency, coverage, and accuracy.

A. Performance Evaluation

This section quantifies the performance gains of applying DuelCeptron to this case study. We evaluate three L2C prefetchers: Signature Path Prefetcher (SPP) [54], Variable Length Delta Prefetcher (VLDP) [55], and Perceptron-based Prefetch Filtering (PPF) [30], similar to the original work.

Figure 16 compares the performance of (i) the composite scheme that uses the context-oblivious SD to dynamically select between the PSA and PSA-2MB versions of the L2C prefetcher (PSA+SD), as presented in the original paper [8], and (ii) the CASD-based composite scheme (PSA+DuelCeptron) that uses DuelCeptron to drive the activation of the PSA and PSA-2MB prefetchers. We perform this comparison across all SPP, VLDP, and PPF prefetchers. The speedups of Figure 16 are computed over the baseline version of the corresponding prefetcher, similar to [8].

Overall, PSA+DuelCeptron outperforms PSA+SD in geomean by 1.5%, 1.6%, and 1.0% across SPP, VLDP, and PPF, respectively. In all three cases, DuelCeptron consistently outperforms SD by leveraging contextual information that enables more informed and timely policy selection. Notably, for workloads with frequent phase changes, where the optimal choice alternates between Pref-PSA and Pref-PSA-2MB, DuelCeptron outperforms original SD. For example, DuelCeptron improves SPP's speedup over original SD by 4.6% and 13.1% for fotonik3d and cvp1_fp15 benchmarks, respectively.

The performance gains of PSA+DuelCeptron do not originate from a single factor but rather from improvements across multiple dimensions such as timeliness, coverage, and accuracy, similar to PSA+SD [8]. Figure 17 justifies this by comparing the impact of SD and DuelCeptron on the coverage, accuracy, and timeliness of the SPP prefetcher across a few representative workloads. We use L2C access latency as a metric to assess the impact on prefetching timeliness, as improved timeliness directly reduces access latency. Figure 17 shows that the speedups of both PSA+SD and PSA+DuelCeptron do not have a single root cause. The dominant source of benefit varies on a workload-by-workload basis. Perceptrons correlate program features with which prefetcher (Pref-PSA vs Pref-PSA-2MB) will be beneficial for the current context, so the selection switches more precisely when needed. This is the reason why looking only at the reported averages across 145 workloads does not provide a clear understanding. For example, PSA+DuelCeptron outperforms PSA+SD for the wrf benchmark mainly due to better prefetching timeliness, while for cvp1_fp15 PSA+DuelCeptron yields better coverage, accuracy, and timeliness than PSA+SD.

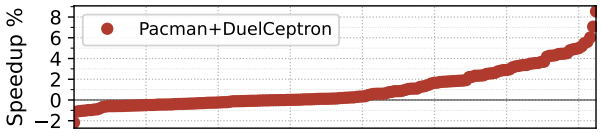

Fig. 18: Performance of Pacman+DuelCeptron over a baseline with Pacman+SD [6] as replacement policy.

VII. Case Study 3: Prefetch-Aware Cache Replacement Policy

To further highlight CASD's effectiveness, we consider as a third case study Pacman [6], a prefetch-aware replacement policy. Pacman uses SD to handle prefetched lines differently from demand lines. This is a slightly different arbitration decision from the previous case studies since Pacman arbitrates how prefetch and demand lines are managed in the cache.

We replace Pacman's context-oblivious SD with DuelCeptron, leaving the underlying policies unchanged, and following the feature selection methodology of Section IV-A3. The selected features are: PC and PC$\oplus$(PA$\gg$12) from Table I. Figure 18 reports the speedups of Pacman+DuelCeptron over a baseline with original Pacman+SD as replacement policy. Overall, Pacman+DuelCeptron improves performance over original Pacman by 1.9% in geomean across the workloads of Section V-A. Consistent with our other case studies, the gains stem from selecting the right policy on a per-context basis, whereas the context-oblivious global counter of the original SD selects a single policy across all requests of a window. This confirms that CASD generalizes beyond a single SD application: the same framework and feature selection process, with no change to the underlying policies, yields consistent benefits across three distinct SD applications.

VIII. Related work

Our proposal, CASD, provides a modular framework that can be seamlessly integrated with SD-based designs to enhance their effectiveness without requiring changes to the underlying policies. As microarchitectural designs increasingly rely on adaptive techniques to sustain performance across diverse workloads, the role of dynamic policy arbitration becomes more important; thus, we anticipate CASD to have a growing impact on next-generation CPU designs.

Cache Replacement Policies. General-purpose policies make decisions either based on line recency, ignoring prior miss history [5], [56]–[60], or using past accesses to estimate reuse distances [31], [53], [61]–[70]. Prefetch-aware policies distinguish between demand and prefetch requests to improve cache management. PACIPV [71] proposes a replacement policy derived from an offline exploration of insertion and promotion policies using different RRPVs [5] for demand and prefetched lines. Another line of work explores code-aware replacement policies, which target workloads with large code footprints and prioritize instruction lines over data lines in lower-level caches [51], [72], [73]. CLIP [51], built on top of SRRIP [5], increases the priority of code lines, Emissary [72] prevents the eviction of performance-critical code lines from the L2C, while Bumper [73] correlates commit information with code line usefulness to keep in the cache only the useful code lines. Recent work explores cooperative TLB and cache policies. Chasapis et al. [74] combine a TLB replacement policy that maximizes the number of instruction hits in the last-level TLB at the expense of increasing data page walks and a complementary L2C policy that accelerates data page walks by prioritizing data translation lines in the L2C over other line types. These works focus on improving replacement decisions within a single policy. Our work is orthogonal, as it improves the arbitration mechanism that selects among competing policies rather than modifying the policies themselves.

Learning-based Predictors for Cache Management. A body of work [30], [31], [53], [62]–[65] predicts a per-line property such as reuse distance for a single mechanism. CASD operates at a different level and is complementary to these predictors. They differ in function: such predictors estimate one quantity for one mechanism, whereas CASD selects which of several competing mechanisms to apply and treats each as a black box. They also differ in role: such predictors are themselves the policy, while CASD modifies no underlying policy and can have such a predictor as one of its competing policies. Our contribution is not the use of perceptrons, explored by prior work [25], [30], [31], [62], but making original SD's arbitration context-aware while preserving its policy-agnostic, Leader-Set training. The distinction is also empirical: although the state-of-the-art reuse predictor Mockingjay [53] is a calibration point, DuelCeptron matches or outperforms it on most studied workloads at lower storage, as shown in Section V-F. DuelCeptron leads where execution context cleanly separates cache-friendly from thrash behavior, Mockingjay where per-line reuse distance dominates, indicating the two capture different signals rather than the same one.

Dead Block Prediction (DBP). Prior art [63], [69], [75]–[84] uses DBP to identify cache lines that will no longer be reused before eviction, thereby improving cache utilization and energy efficiency. Early work introduced counter-based and signature-based predictors that track block reuse patterns to decide when a line becomes dead [69], [75]. Later studies proposed sampling-based and history-based methods to reduce overheads while maintaining accuracy [63]. DBP has been applied in various contexts, including cache replacement [31], bypassing [63], and power reduction [79]. Despite extensive research, accurately predicting dead blocks remains challenging due to dynamic program behavior and complex access patterns. While DBP improves replacement decisions by predicting reuse at the block level, CASD complements DBP-based techniques by operating at a higher level and selecting between competing policies based on execution context.

Hardware Prefetching. Spatio-temporal prefetchers rely on the premise that temporally or spatially adjacent accesses repeat [13], [14], [33]–[35], [46], [54], [55], [85]–[94]. Other prefetchers target irregular memory accesses [95]–[100]. Runahead prefetchers speculatively execute address-generation code to prefetch accesses beyond the visibility of the OoO window [101]–[106]. Our work is orthogonal to hardware prefetching as it focuses on arbitration between competing schemes (*e.g.*, prefetch engines as in Section VI).

IX. Conclusions

This work provides evidence that retrofitting contextual learning into dynamic policy arbitration significantly improves decision quality over traditional context-oblivious approaches such as Set Dueling (SD). We propose *Context-Aware Set Dueling (CASD)*, a general and holistic framework that enhances SD with a predictive, context-aware mechanism that learns correlations between execution context and policy effectiveness, enabling accurate fine-grained policy selection with minimal storage and complexity overheads.

We demonstrate the benefits of CASD through *DuelCeptron*, a perceptron-based instantiation of CASD evaluated across three case studies and a diverse set of workloads. DuelCeptron consistently outperforms conventional SD by localizing policy selection to the contexts where each policy is most effective.

Workload diversity is pushing adaptive techniques into every corner of CPU design, making the quality of the arbitration among them a first-order concern. CASD offers a practical and extensible path: it evolves SD, a mechanism that has influenced commercial CPU designs, at kilobyte-scale cost and without modifying the policies it arbitrates. *DuelCeptron* makes two deliberately conservative choices: domain-agnostic features and a perceptron-based decision logic. Specializing the features per application domain, adopting a more expressive learning model, or extending CASD beyond two competing policies all remain open research questions.

References

[1] "Hot chips 2023: Arm's neoverse V2," https://chipsandcheese.com/2023/09/11/hot-chips-2023-arms-neoverse-v2/, 2023, accessed: 2026-08-24.

[2] "Arm Neoverse V2," https://www.arm.com/products/silicon-ip-cpu/neoverse/neoverse-v2, 2022, accessed: 2026-08-24.

[3] P. J. Moyer, "Scaled Set Dueling for Cache Replacement Policies," U.S. Patent 10,430,349 B2, 2019, assignee: Advanced Micro Devices, Inc.

[4] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," *SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 381–391, 2007. [Online]. Available: https://doi.org/10.1145/1273440.1250709

[5] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 60–71. [Online]. Available: https://doi.org/10.1145/1815961.1815971

[6] C.-J. Wu, A. Jaleel, M. Martonosi, S. C. Steely, and J. Emer, "PACMan: Prefetch-aware cache management for high performance caching," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO-44. New York, NY, USA: Association for Computing Machinery, 2011, pp. 442–453. [Online]. Available: https://doi.org/10.1145/2155620.2155672

[7] K. Weston, V. Janfaza, A. Johnson, and A. Muzahid, "A cost-effective dueling framework for set-associative cache indexing," in *Proceedings of the 39th ACM International Conference on Supercomputing*, ser. ICS '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 1176–1189. [Online]. Available: https://doi.org/10.1145/3721145.3729513

[8] G. Vavouliotis, G. Chacon, L. Alvarez, P. V. Gratz, D. A. Jiménez, and M. Casas, "Page size aware cache prefetching," in *Proceedings of the 55th International Symposium on Microarchitecture*, ser. MICRO '22. IEEE, 2022, pp. 956–974. [Online]. Available: https://doi.org/10.1109/MICRO56248.2022.00070

[9] "Selective control flow predictor insertion," U.S. Patent Application 2024/0 095 034 A1, 2024, describes a set-dueling heuristic for control-flow predictor insertion. [Online]. Available: https://patents.google.com/patent/US20240095034A1/en

[10] S. Briongos, P. Malagón, J. M. Moya, and T. Eisenbarth, "RELOAD+REFRESH: Abusing cache replacement policies to perform stealthy cache attacks," in *29th USENIX Security Symposium (USENIX Security 20)*. Boston, MA, USA: USENIX Association, Aug. 2020, pp. 1967–1984. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/briongos

[11] P. Vila, P. Ganty, M. Guarnieri, and B. Köpf, "CacheQuery: Learning replacement policies from hardware caches," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 519–532. [Online]. Available: https://doi.org/10.1145/3385412.3386008

[12] "4th data prefetching championship 2026," https://sites.google.com/view/dpc4-2026/, 2026, accessed: 2026-08-24.

[13] A. Navarro-Torres, B. Panda, J. Alastruey-Benedé, P. Ibáñez, V. Viñals-Yúfera, and A. Ros, "Berti: an accurate local-delta data prefetcher," in *Proceedings of the 55th International Symposium on Microarchitecture*, ser. MICRO '22. IEEE, 2022, pp. 975–991. [Online]. Available: https://doi.org/10.1109/MICRO56248.2022.00072

[14] S. Somogyi, T. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *Proceedings of the 33rd International Symposium on Computer Architecture*, ser. ISCA '06. IEEE, 2006, pp. 252–263. [Online]. Available: https://doi.org/10.1109/ISCA.2006.38

[15] "SPEC CPU 2006," https://www.spec.org/cpu2006/, 2006, accessed: 2026-08-24.

[16] "SPEC CPU 2017," https://www.spec.org/cpu2017/, 2017, accessed: 2026-08-24.

[17] S. Beamer, K. Asanovic, and D. A. Patterson, "The GAP benchmark suite," arXiv:1508.03619 [cs.DC], 2015. [Online]. Available: http://arxiv.org/abs/1508.03619

[18] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 135–146. [Online]. Available: https://doi.org/10.1145/2442516.2442530

[19] "Championship value prediction (CVP)," https://www.microarch.org/cvp1/, accessed: 2026-08-24.

[20] J. Feliu, A. Perais, D. Jimenez, and A. Ros, "Rebasing microarchitectural research with industry traces," in *2023 IEEE International Symposium on Workload Characterization (IISWC)*. Ghent, Belgium: IEEE Computer Society, 2023, pp. 100–114. [Online]. Available: https://doi.org/10.1109/IISWC59245.2023.00027

[21] D. Patsidis and G. Vavouliotis, "Context-aware set dueling for dynamic policy arbitration," *IEEE Computer Architecture Letters*, vol. 24, no. 2, pp. 301–304, 2025. [Online]. Available: https://doi.org/10.1109/LCA.2025.3617159

[22] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Computer Architecture News*, vol. 23, no. 1, pp. 20–24, 1995. [Online]. Available: https://doi.org/10.1145/216585.216588

[23] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*, ser. CF '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 162. [Online]. Available: https://doi.org/10.1145/977091.977115

[24] F. Rosenblatt, "The Perceptron: A Perceiving and Recognizing Automaton (Project PARA)," Cornell Aeronautical Laboratory, Buffalo, NY, USA, Tech. Rep. 85-460-1, 1957.

[25] D. Jiménez and C. Lin, "Dynamic Branch Prediction with Perceptrons," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, ser. HPCA '01, 2001, pp. 197–206. [Online]. Available: https://doi.org/10.1109/HPCA.2001.903263

[26] 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," arXiv:2210.14324 [cs.AR], 2022. [Online]. Available: https://doi.org/10.48550/ARXIV.2210.14324

[27] "ChampSim," https://github.com/ChampSim/ChampSim, accessed: 2026-08-24.

[28] D. Tarjan and K. Skadron, "Merging path and gshare indexing in perceptron branch prediction," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 3, pp. 280–300, 2005. [Online]. Available: https://doi.org/10.1145/1089008.1089011

[29] G. Vavouliotis, M. Torrents, B. Grot, K. Kalaitzidis, L. Peled, and M. Casas, "To cross, or not to cross pages for prefetching?" in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, ser. HPCA '25. IEEE, 2025, pp. 188–203. [Online]. Available: https://doi.org/10.1109/HPCA61900.2025.00025

[30] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–13. [Online]. Available: https://doi.org/10.1145/3307650.3322207

[31] D. A. Jiménez and E. Teran, "Multiperspective reuse prediction," in *Proceedings of the 50th International Symposium on Microarchitecture*, ser. MICRO '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 436–448. [Online]. Available: https://doi.org/10.1145/3123939.3123942

[32] G. Vavouliotis, "Agentic LLMs for microarchitecture research: The role of the human architect," *IEEE Computer Architecture Letters*, vol. 25, no. 2, pp. 299–302, 2026. [Online]. Available: https://doi.org/10.1109/LCA.2026.3718795

[33] G. Gerogiannis and J. Torrellas, "Micro-armed bandit: Lightweight & reusable reinforcement learning for microarchitecture decision-making," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 698–713. [Online]. Available: https://doi.org/10.1145/3613424.3623780

[34] C. Block, G. Gerogiannis, and J. Torrellas, "Micro-MAMA: Multi-agent reinforcement learning for multicore prefetching," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 884–898. [Online]. Available: https://doi.org/10.1145/3725843.3756096

[35] R. Bera, K. Kanellopoulos, A. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A customizable hardware prefetching framework using online reinforcement learning," in *Proceedings of the 54th*

International Symposium on Microarchitecture, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1121–1137. [Online]. Available: https://doi.org/10.1145/3466752.3480114

[36] A. V. Jamet, G. Vavouliotis, M. Torrents, D. Chasapis, and M. Casas, "Enhancing instruction prefetching via cache and TLB management," in *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2026, pp. 2254–2269. [Online]. Available: https://doi.org/10.1109/ISCA66397.2026.00159

[37] K. Weston, A. Johnson, V. Janfaza, F. Mahmud, and A. Muzahid, "Customizing cache indexing through entropy estimation," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO '24. IEEE, 2024, pp. 451–463. [Online]. Available: https://doi.org/10.1109/MICRO61859.2024.00041

[38] G. Reinman, B. Calder, and T. Austin, "Fetch Directed Instruction Prefetching," in *Proceedings of the 32nd International Symposium on Microarchitecture*, ser. MICRO '99. IEEE, 1999, pp. 16–27. [Online]. Available: https://doi.org/10.1109/MICRO.1999.809439

[39] A. Bhattacharjee and D. Lustig, *Architectural and Operating System Support for Virtual Memory*. Morgan & Claypool Publishers, 2017. [Online]. Available: https://doi.org/10.1007/978-3-031-01757-5

[40] J. L. Hennessy, D. A. Patterson, and C. Kozyrakis, *Computer Architecture: A Quantitative Approach*, 7th ed. Cambridge, MA, USA: Morgan Kaufmann, 2025.

[41] A. Bhattacharjee, "Large-reach memory management unit caches," in *Proceedings of the 46th International Symposium on Microarchitecture*, ser. MICRO '13. New York, NY, USA: ACM, 2013, pp. 383–394. [Online]. Available: https://doi.org/10.1145/2540708.2540741

[42] D. Schall, A. Sandberg, and B. Grot, "The last-level branch predictor," in *Proceedings of the 57th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '24. IEEE, 2024, pp. 464–479. [Online]. Available: https://doi.org/10.1109/micro61859.2024.00042

[43] M. Carpen-Amarie, G. Vavouliotis, K. Tovletoglou, B. Grot, and R. Mueller, "Concurrent GCs and modern java workloads: A cache perspective," in *Proceedings of the 2023 ACM SIGPLAN International Symposium on Memory Management*, ser. ISMM 2023. New York, NY, USA: Association for Computing Machinery, 2023, pp. 71–84. [Online]. Available: https://doi.org/10.1145/3591195.3595269

[44] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 318–319, 2003. [Online]. Available: https://doi.org/10.1145/885651.781076

[45] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *Proceedings of the 2020 47th International Symposium on Computer Architecture*, ser. ISCA '20. IEEE, 2020, pp. 118–131. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00021

[46] P. Michaud, "Best-offset hardware prefetching," in *Proceedings of the 22nd International Symposium on High Performance Computer Architecture*, ser. HPCA '16. IEEE, 2016, pp. 469–480. [Online]. Available: https://doi.org/10.1109/HPCA.2016.7446087

[47] G. Vavouliotis, L. Alvarez, B. Grot, D. Jiménez, and M. Casas, "Morrigan: A composite instruction TLB prefetcher," in *Proceedings of the 54th International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1138–1153. [Online]. Available: https://doi.org/10.1145/3466752.3480049

[48] G. Vavouliotis, L. Alvarez, V. Karakostas, K. Nikas, N. Koziris, D. A. Jiménez, and M. Casas, "Exploiting page table locality for agile TLB prefetching," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, ser. ISCA '21. IEEE, 2021, pp. 85–98. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00016

[49] G. Vavouliotis, "Advanced hardware prefetching in virtual memory systems," Ph.D. dissertation, Universitat Politècnica de Catalunya (UPC), 2023. [Online]. Available: https://upcommons.upc.edu/entities/publication/f16d637b-ad15-4f69-b830-3471b7f2fb84

[50] A. V. Jamet, G. Vavouliotis, D. A. Jiménez, L. Alvarez, and M. Casas, "A two level neural approach combining off-chip prediction with adaptive prefetch filtering," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, March 2024, pp. 528–542. [Online]. Available: https://doi.org/10.1109/HPCA57654.2024.00046

[51] A. Jaleel, J. Nuzman, A. Moga, S. C. Steely, and J. Emer, "High performing cache hierarchies for server workloads: Relaxing inclusion to capture the latency benefits of exclusive caches," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, ser. HPCA '15. IEEE, 2015, pp. 343–353. [Online]. Available: https://doi.org/10.1109/HPCA.2015.7056045

[52] A. V. Jamet, G. Vavouliotis, D. A. Jiménez, L. Alvarez, and M. Casas, "Practically tackling memory bottlenecks of graph-processing workloads," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024, pp. 1034–1045. [Online]. Available: https://doi.org/10.1109/IPDPS57955.2024.00096

[53] I. Shah, A. Jain, and C. Lin, "Effective mimicry of belady's MIN policy," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, ser. HPCA '22. IEEE, 2022, pp. 558–572. [Online]. Available: https://doi.org/10.1109/HPCA53966.2022.00048

[54] J. Kim, S. H. Pugsley, P. V. Gratz, A. L. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *Proceedings of the 49th International Symposium on Microarchitecture*, ser. MICRO '16. IEEE, 2016, pp. 1–12. [Online]. Available: https://doi.org/10.1109/MICRO.2016.7783763

[55] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 141–152. [Online]. Available: https://doi.org/10.1145/2830772.2830793

[56] H. Gao and C. Wilkerson, "A dueling segmented LRU replacement algorithm with adaptive bypassing," in *1st JILP Workshop on Computer Architecture Competitions (JWAC-1): Cache Replacement Championship*, 2010. [Online]. Available: https://jilp.org/jwac-1/online/papers/005_gao.pdf

[57] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies," in *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '99. New York, NY, USA: Association for Computing Machinery, 1999, pp. 134–143. [Online]. Available: https://doi.org/10.1145/301453.301487

[58] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93. New York, NY, USA: Association for Computing Machinery, 1993, pp. 297–306. [Online]. Available: https://doi.org/10.1145/170035.170081

[59] R. Subramanian, Y. Smaragdakis, and G. H. Loh, "Adaptive caches: Effective shaping of cache behavior to workloads," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, ser. MICRO '06. IEEE, 2006, pp. 385–396. [Online]. Available: https://doi.org/10.1109/MICRO.2006.7

[60] W. Wong and J.-L. Baer, "Modified LRU policies for improving second-level cache behavior," in *Proceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No.PR00550)*, ser. HPCA '00. IEEE, 2000, pp. 49–60. [Online]. Available: https://doi.org/10.1109/HPCA.2000.824338

[61] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely, and J. Emer, "SHiP: signature-based hit predictor for high performance caching," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: Association for Computing Machinery, 2011, pp. 430–441. [Online]. Available: https://doi.org/10.1145/2155620.2155671

[62] E. Teran, Z. Wang, and D. A. Jiménez, "Perceptron learning for reuse prediction," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO-49. Washington, DC, USA: IEEE Computer Society, 2016, pp. 1–12. [Online]. Available: https://doi.org/10.1109/MICRO.2016.7783705

[63] S. M. Khan, Y. Tian, and D. A. Jiménez, "Sampling dead block prediction for last-level caches," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-43. IEEE, 2010, pp. 175–186. [Online]. Available: https://doi.org/10.1109/MICRO.2010.24

[64] Z. Shi, X. Huang, A. Jain, and C. Lin, "Applying deep learning to the cache replacement problem," in *Proceedings*

of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, pp. 413–425. [Online]. Available: https://doi.org/10.1145/3352460.3358319

[65] A. Jain and C. Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, ser. ISCA '16. IEEE, 2016, pp. 78–89. [Online]. Available: https://doi.org/10.1109/ISCA.2016.17

[66] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. USA: IEEE Computer Society, 2012, pp. 389–400. [Online]. Available: https://doi.org/10.1109/MICRO.2012.43

[67] Z. Hu, S. Kaxiras, and M. Martonosi, "Timekeeping in the memory system: Predicting and optimizing memory behavior," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ser. ISCA '02. USA: IEEE Computer Society, 2002, pp. 209–220. [Online]. Available: https://doi.org/10.1109/ISCA.2002.1003579

[68] G. Keramidas, P. Petoumenos, and S. Kaxiras, "Cache replacement based on reuse-distance prediction," in *2007 25th International Conference on Computer Design*. IEEE, 2007, pp. 245–250. [Online]. Available: https://doi.org/10.1109/ICCD.2007.4601909

[69] M. Kharbutli and Y. Solihin, "Counter-based cache replacement and bypassing algorithms," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 433–447, 2008. [Online]. Available: https://doi.org/10.1109/TC.2007.70816

[70] Sweta, P. Priyadarshini, and B. Panda, "Drishti: Do not forget slicing while designing last-level cache replacement policies for many-core systems," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 519–532. [Online]. Available: https://doi.org/10.1145/3725843.3756028

[71] 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*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 1005–1019. [Online]. Available: https://doi.org/10.1145/3695053.3730993

[72] N. P. Nagendra, B. R. Godala, I. Chaturvedi, A. Patel, S. Kanev, T. Moseley, J. Stark, G. A. Pokam, S. Campanoni, and D. I. August, "Emissary: Enhanced miss awareness replacement policy for L2 instruction caching," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589097

[73] G. Vavouliotis, T. Rollet, D. B. Bartolini, B. Grot, L. Peled, and L. Yang, "Bumper: Hinting instruction usefulness for robust unified caches," in *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*, 2026, pp. 2029–2045. [Online]. Available: https://doi.org/10.1109/ISCA66397.2026.00145

[74] D. Chasapis, G. Vavouliotis, D. A. Jiménez, and M. Casas, "Instruction-aware cooperative TLB and cache replacement policies," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 619–636. [Online]. Available: https://doi.org/10.1145/3669940.3707247

[75] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-block prediction and dead-block correlating prefetchers," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ser. ISCA '01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 144–154. [Online]. Available: https://doi.org/10.1145/379240.379259

[76] C.-H. Chi and H. Dietz, "Improving cache performance by selective cache bypass," in *[1989] Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume 1: Architecture Track*, vol. 1. IEEE, 1989, pp. 277–285. [Online]. Available: https://doi.org/10.1109/HICSS.1989.47168

[77] T. L. Johnson, D. A. Connors, M. C. Merten, and W.-m. W. Hwu, "Run-time cache bypassing," *IEEE Trans. Comput.*, vol. 48, no. 12, pp. 1338–1354, 1999. [Online]. Available: https://doi.org/10.1109/12.817393

[78] E. Tam, J. Rivers, V. Srinivasan, G. Tyson, and E. Davidson, "Active management of data caches by exploiting reuse information," *IEEE Transactions on Computers*, vol. 48, no. 11, pp. 1244–1259, 1999. [Online]. Available: https://doi.org/10.1109/12.811113

[79] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, ser. ISCA '01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 240–251. [Online]. Available: https://doi.org/10.1145/379240.379268

[80] J. Jalminger and P. Stenstrom, "A novel approach to cache block reuse predictions," in *2003 International Conference on Parallel Processing, 2003. Proceedings.* IEEE, 2003, pp. 294–302. [Online]. Available: https://doi.org/10.1109/ICPP.2003.1240592

[81] H. Dybdahl and P. Stenström, "Enhancing last-level cache performance by block bypassing and early miss determination," in *Proceedings of the 11th Asia-Pacific Conference on Advances in Computer Systems Architecture*, ser. ACSAC'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 52–66. [Online]. Available: https://doi.org/10.1007/11859802_6

[82] A. González, C. Aliagas, and M. Valero, "A data cache with multiple caching strategies tuned to different types of locality," in *Proceedings of the 9th International Conference on Supercomputing*, ser. ICS '95. New York, NY, USA: Association for Computing Machinery, 1995, pp. 338–347. [Online]. Available: https://doi.org/10.1145/224538.224622

[83] H. Liu, M. Ferdman, J. Huh, and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '08. IEEE, 2008, pp. 222–233. [Online]. Available: https://doi.org/10.1109/MICRO.2008.4771793

[84] P. Faldu and B. Grot, "Leeway: Addressing variability in dead-block prediction for last-level caches," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, ser. PACT '17. IEEE, 2017, pp. 180–193. [Online]. Available: https://doi.org/10.1109/PACT.2017.32

[85] B. Falsafi and T. F. Wenisch, *A Primer on Hardware Prefetching*. Morgan & Claypool Publishers, 2014. [Online]. Available: https://doi.org/10.1007/978-3-031-01743-8

[86] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *Proceedings of the 25th International Symposium on High Performance Computer Architecture*, ser. HPCA '19. IEEE, 2019, pp. 399–411. [Online]. Available: https://doi.org/10.1109/HPCA.2019.00053

[87] H. Wu, K. Nathella, J. Pusdesris, D. Sunwoo, A. Jain, and C. Lin, "Temporal prefetching without the off-chip metadata," in *Proceedings of the 52nd International Symposium on Microarchitecture*, ser. MICRO '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 996–1008. [Online]. Available: https://doi.org/10.1145/3352460.3358300

[88] S. Ainsworth and L. Mukhanov, "Triangel: A high-performance, accurate, timely on-chip temporal prefetcher," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, ser. ISCA '24. Los Alamitos, CA, USA: IEEE Computer Society, 2024, pp. 1202–1216. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00090

[89] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos, "Practical off-chip meta-data for temporal memory streaming," in *Proceedings of the 15th International Symposium on High Performance Computer Architecture*, ser. HPCA '09. IEEE, 2009, pp. 79–90. [Online]. Available: https://doi.org/10.1109/hpca.2009.4798239

[90] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Domino temporal data prefetcher," in *Proceedings of the 24th International Symposium on High Performance Computer Architecture*, ser. HPCA '18. IEEE, 2018, pp. 131–142. [Online]. Available: https://doi.org/10.1109/HPCA.2018.00021

[91] K. Nesbit and J. Smith, "Data cache prefetching using a global history buffer," in *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, ser. HPCA '04. IEEE, 2004, p. 96. [Online]. Available: https://doi.org/10.1109/HPCA.2004.10030

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

[93] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," in *Proceedings of the*

25th International Symposium on Computer Architecture, ser. ISCA '98. IEEE, 1998, pp. 357–368. [Online]. Available: https://doi.org/10.1109/ISCA.1998.694794

[94] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *Proceedings of the 46th International Symposium on Microarchitecture*, ser. MICRO '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 247–259. [Online]. Available: https://doi.org/10.1145/2540708.2540730

[95] M. Cavus, R. Sendag, and J. J. Yi, "Informed prefetching for indirect memory accesses," *ACM Trans. Archit. Code Optim.*, vol. 17, no. 1, 2020. [Online]. Available: https://doi.org/10.1145/3374216

[96] F. Xue, C. Han, X. Li, J. Wu, T. Zhang, T. Liu, Y. Hao, Z. Du, Q. Guo, and F. Zhang, "Tyche: An efficient and general prefetcher for indirect memory accesses," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 2, 2024. [Online]. Available: https://doi.org/10.1145/3641853

[97] A. Roth, A. Moshovos, and G. S. Sohi, "Dependence based prefetching for linked data structures," in *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '98. New York, NY, USA: Association for Computing Machinery, 1998, pp. 115–126. [Online]. Available: https://doi.org/10.1145/291069.291034

[98] A. Roth and G. S. Sohi, "Effective jump-pointer prefetching for linked data structures," in *Proceedings of the 26th International Symposium on Computer Architecture*, ser. ISCA '99. USA: IEEE Computer Society, 1999, pp. 111–121. [Online]. Available: https://doi.org/10.1145/307338.300989

[99] M. Bekerman, S. Jourdan, R. Ronen, G. Kirshenboim, L. Rappoport, A. Yoaz, and U. Weiser, "Correlated load-address predictors," in *Proceedings of the 26th International Symposium on Computer Architecture*, ser. ISCA '99. New York, NY, USA: Association for Computing Machinery, 1999, pp. 54–63. [Online]. Available: https://doi.org/10.1145/307338.300984

[100] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "IMP: Indirect memory prefetcher," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 178–190. [Online]. Available: https://doi.org/10.1145/2830772.2830807

[101] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt, "Runahead execution: an alternative to very large instruction windows for out-of-order processors," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, ser. HPCA '03. IEEE, 2003, pp. 129–140. [Online]. Available: https://doi.org/10.1109/HPCA.2003.1183532

[102] M. Hashemi and Y. N. Patt, "Filtered runahead execution with a runahead buffer," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 358–369. [Online]. Available: https://doi.org/10.1145/2830772.2830812

[103] M. Hashemi, O. Mutlu, and Y. N. Patt, "Continuous runahead: Transparent hardware acceleration for memory intensive workloads," in *Proceedings of the 49th International Symposium on Microarchitecture*, ser. MICRO-49. IEEE, 2016, pp. 1–12. [Online]. Available: https://doi.org/10.1109/MICRO.2016.7783764

[104] A. Naithani, J. Feliu, A. Adileh, and L. Eeckhout, "Precise runahead execution," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, ser. HPCA '20. IEEE, 2020, pp. 397–410. [Online]. Available: https://doi.org/10.1109/HPCA47549.2020.00040

[105] A. Naithani, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Vector runahead," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, ser. ISCA '21. IEEE, 2021, pp. 195–208. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00024

[106] A. Naithani, J. Roelandts, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Decoupled vector runahead," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 17–31. [Online]. Available: https://doi.org/10.1145/3613424.3614255