2026 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

Streamlined On-Chip Temporal Prefetching

Quang Duong and Calvin Lin
Department of Computer Science, The University of Texas at Austin
{qduong,lin}@cs.utexas.edu

***Abstract*—In this paper, we present the Streamline temporal prefetcher, which introduces a stream-based metadata representation that produces three significant benefits over Triangel, the previous state-of-the-art in temporal prefetching. First, it removes redundancy present in Triangel's metadata. Second, it prioritizes the storage of those metadata entries that have higher prefetch utility. Third, it eliminates the need for the untenable LLC traffic that is induced when Triangel dynamically adjusts the size of its metadata store. The end result is that for memory-intensive SPEC 2006, SPEC 2017, and GAP benchmarks, Streamline outperforms Triangel by 6.7 percentage points on an 8-core system with a stride prefetcher. This performance benefit comes from Streamline's improved storage efficiency: It holds 33% more correlations than Triangel, which translates to 12.5 percentage points better prefetch coverage. Significantly, Streamline matches the performance of Triangel even when Triangel is given twice the metadata storage.**

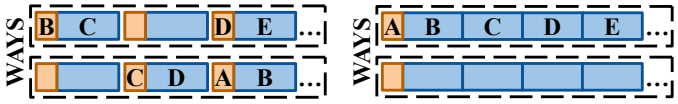
I. Introduction

Temporal prefetching, also known as correlation-based prefetching, is an important form of prefetching because it can cover irregular memory access streams. The idea is to learn pairs of correlated addresses: If address X is typically followed by address Y, then X can serve as a trigger for the prefetch of Y. Thus, temporal prefetchers can prefetch arbitrary repeated sequences of addresses. However, since these prefetchers essentially memorize correlations, they can require huge amounts of metadata, so work in this area has focused on the management of metadata.

Early work stored the metadata off chip and employed techniques to amortize the cost of their access and to probabilistically omit their update [51]. Later work introduced a metadata representation that allowed it to be cached [20] and prefetched [55].

A significant milestone was the advent of *on-chip* temporal prefetchers. The Triage prefetcher [54] stores in a partition of the LLC only the most frequently accessed portion of its metadata and discards the rest. Arm produced a commercial on-chip temporal prefetcher in 2020 [29]–[31], [40], and recently, Ainsworth and Mukhanov [4] provide important implementation details that were missing from the Triage paper. Their Triangel prefetcher improves upon Triage by performing accuracy-based metadata filtering and degree control.

Despite these advances, on-chip temporal prefetchers face three challenges. First, to increase prefetch coverage, they need to store as many useful correlations as possible. Second, because the metadata store competes with the data partition for LLC real estate, they need to minimize the size of the metadata partition. And third, they require a metadata organization that avoids the high cost of dynamically resizing the metadata partition. In this paper, we address all three issues by leveraging a simple design principle for metadata management: *Exploit the structure and semantics of streams.* This principle has significant implications, as we now explain.



(a) **Pairwise Metadata** (b) **Stream-Based Metadata**

Fig. 1: **Benefits of Stream-Based Designs**: We show two truncated LLC ways of metadata entries for the stream [A, B, C, D, E]. **(a)** Pairwise entries redundantly store addresses as both **trigger** and **prefetch target**. Moreover, since they're inserted independently, temporal adjacency doesn't always translate to spatial locality. **(b)** Stream-based entries hold 33% more correlations and exhibit good spatial locality.

First, because temporal metadata describes sequences—or streams—of data addresses, our prefetcher uses a stream-based metadata format, which produces three benefits. (1) *Our stream-based format stores metadata more compactly than in prior work.* Figure 1a shows how the previously-used pairwise format stores all but the first and last address in a stream twice, once as the trigger and then again as the prefetch target. By contrast, Figure 1b shows how our stream-based format eliminates this redundancy, enabling the storage of 33% more correlations. (2) *Our stream-based format has better spatial locality and induces less metadata traffic for higher degree prefetches.* Figure 1 shows that a pairwise format has its correlations scattered across the metadata store, so it requires multiple metadata reads to issue higher-degree prefetches, whereas our stream-based format requires just one. (3) *Our stream-based format fixes a fundamental issue in the design of Triangel*: When Triangel's metadata store is resized, the metadata index function changes, leaving metadata entries misplaced (see Section III-C2 for details). Thus, to resize its metadata store, Triangel must shuffle up to 1MB (average of 516KB) of metadata to move them to the correct locations, incurring significant metadata traffic to the LLC. Our stream-based format enables a new indexing scheme that prevents metadata entries from being misplaced, eliminating the need for a costly shuffling operation.

Second, our prefetcher uses the *prefetch utility* of the metadata when deciding which metadata to store. Both Triage and Triangel view metadata as raw data, so they store triggers that appear most frequently, even if those triggers do not lead

to useful prefetches. By contrast, the consideration of prefetch utility enables our prefetcher to prioritize the storage of entries that lead to useful prefetches.

Unfortunately, a stream-based representation also introduces three adverse side effects: (1) reduced prefetch coverage because the reduced number of triggers increases the likelihood of missing a trigger, (2) a new form of redundancy when metadata entries overlap, and (3) increased conflict misses because the larger metadata entry size reduces associativity. This paper resolves each of these problems, enabling the design of a new on-chip temporal prefetcher that uses a stream-based representation to significantly improve storage efficiency and thus improve overall performance.

This paper makes the following contributions:

- We present Streamline, an on-chip temporal prefetcher that uses a stream-based metadata representation to provide several benefits over Triangel:
  - *Storage Efficiency*. For the same metadata store size, Streamline holds 33% more correlations, which leads to an increase in prefetch coverage by 12.5 percentage points. Moreover, (1) Streamline can outperform Triangel even when Triangel is given twice as much metadata storage (1MB vs. 0.5MB), and (2) when both prefetchers are given 1MB metadata stores, Streamline can outperform Triangel even when Triangel's metadata is placed in a separate *dedicated store.*
  - *Performance*. On an 8-core system with an L1D stride prefetcher, Streamline outperforms Triangel by 6.7 percentage points. On a single-core system with an L1D stride prefetcher, Streamline outperforms Triangel by 3 percentage points, and Streamline's accuracy is greater by 3.6 percentage points.
  - *Bandwidth Efficiency*. Streamline's compact metadata format also improves bandwidth efficiency: Streamline generates 39% to 87% less metadata traffic to the LLC.
  - *Simplicity*. Streamline eliminates the need for Triangel's costly metadata rearrangement operation.
- We incorporate prefetch utility into Streamline's metadata management policies: We (1) reformulate Belady's MIN [9] for temporal metadata and (2) introduce *Utility-Aware Dynamic Partitioning*, which utilizes prefetch accuracy to better size the metadata partition than Triangel.

The remainder of the paper is organized as follows. We first contextualize our work in Section II and then provide relevant background in Section III. Next, we describe our solution in Section IV and then present our empirical evaluation in Section V. Finally, we conclude in Section VI.

II. Related Work

We now place our work in the context of previous work.

A. Temporal Prefetchers

Joseph and Grunwald [26] introduce address correlation to target temporal locality. By learning correlations, temporal prefetchers are quite powerful since they can cover any repeated memory sequence. Early temporal prefetchers [26], [36] store their correlations in tiny on-chip structures, but their limited capacity fails to capture enough of the access stream to see significant performance, demonstrating the high storage costs needed for effective temporal prefetching.

Thus, the majority of temporal prefetching research focuses on efficiently managing this metadata by storing the bulk of it off chip [6], [20], [49], [51], [52], [55] and fetching the metadata on demand to issue prefetches. Recent work [4], [53], [54] eliminates this additional off-chip traffic by storing the metadata in a partition of the LLC. We now discuss prior work in temporal prefetching in more detail.

1) Off-Chip Temporal Prefetchers: Nesbit, et al. [36] introduce the Global History Buffer (GHB), which is the basis of early work in off-chip temporal prefetching. The GHB stores accesses in a buffer and uses an index table to search for delta and address correlations. For the large metadata requirements of temporal prefetching, the GHB resides in DRAM [6], [20], [51], [52], [55], leading to significant DRAM traffic to access prefetcher metadata.

Wenisch, et al.'s STMS prefetcher [51] reduces metadata traffic by amortizing the cost of writes via probabilistic sampling, and it amortizes the cost of reads by fetching long streams of metadata. Bakhshalipour, et al. [6] enhance the index table to halve the metadata traffic required to use longer history lengths.

Jain, et al. introduce the ISB [20], which uses a structural address space to enable PC-localized metadata to be cached on chip, but the ISB suffers from poor metadata utilization. Wu, et al.'s MISB [55] prefetches metadata within the structural address space to improve timeliness and utilization, but MISB still incurs significant off-chip traffic.

Streamline differs from this body of work because it stores its metadata on chip, so it incurs no off-chip metadata traffic. And unlike the GHB, which is also a stream-based representation, Streamline's representation does not redundantly store a separate entry for each memory access.

2) On-Chip Temporal Prefetchers: Triage [53], [54] keeps its metadata on chip to eliminate this costly off-chip traffic, and Triangel [4] improves Triage by increasing prefetch accuracy and reducing on-chip traffic (see Section III for a more detailed description of both prefetchers). Our work improves performance by better managing this on-chip metadata store.

B. Other Prefetching Work

There exists decades of research in data prefetching that targets different patterns such as deltas [11], [14], [17], [19], [24], [27], [32], [35], [37]–[39], [41], [46], [47], [57], spatial footprints [7], [12], [50], pointer chases [15], [44], and strided indirect array accesses [56]. Whereas regular prefetchers can cover some irregular patterns, temporal prefetchers can do so more accurately, which is critical for bandwidth-constrained multi-core environments (see Figure 11d). Another line of work instead uses neural models to perform temporal prefetching [16], [48], which focuses on improving the prediction mechanism rather than the management of the metadata store.

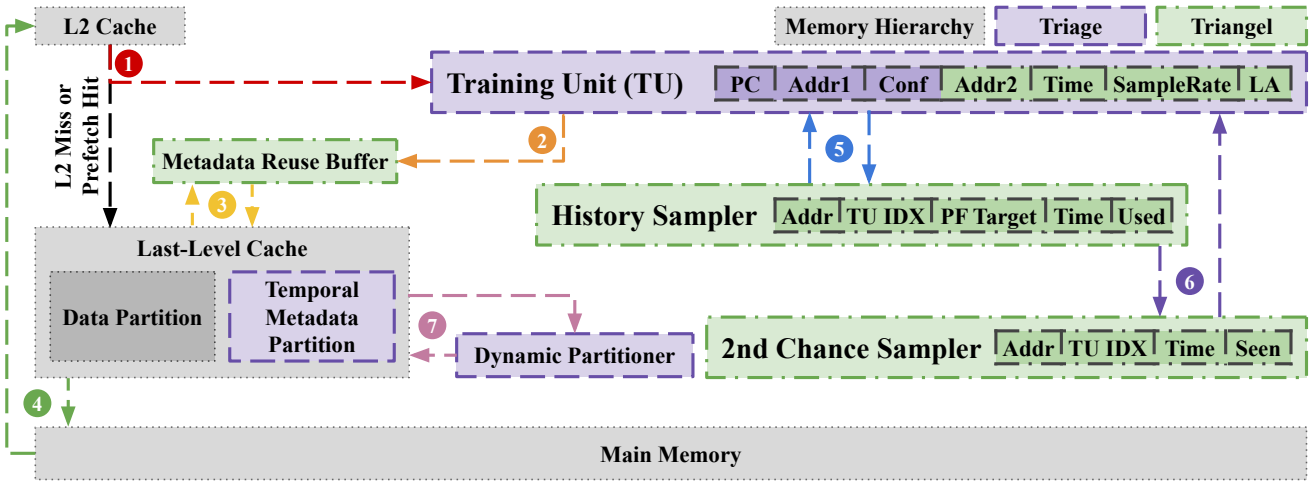


Fig. 2: **Overview of Triage and Triangel**

In addition, there is extensive work in mitigating load latencies through techniques such as software prefetching [5], [23], [58], criticality-based scheduling [28], runahead execution [33], [34], [43], off-chip load prediction [10], and load elimination [13], but these are orthogonal to our approach.

III. Background

In this section, we first provide background on Triage [54] and Triangel [4] that is helpful for understanding the rest of this paper. Section III-C then describes the issues with these prior approaches.

A. Triage

Triage [54] is the first temporal prefetcher to store its metadata in a partition of the LLC; it discards any metadata that doesn't fit. Triage assumes that the LLC is way-partitioned so that in each set some ways are allocated for metadata and the rest for data. Triage sizes this partition to maximize the trigger hit rate. Triage's training unit (TU) stores each PC's last access to be the trigger for its next access.

1) Hardware Design: Figure 2 shows Triage's overall design. Its metadata entries consist of a hashed trigger address (10 bits), a compressed prefetch target (10-bit lookup table (LUT) index and 11-bit tag), and a confidence bit, so 16 entries fit in a block. While LUT compression increases the storage capacity, Triangel's authors show that it significantly reduces prefetch accuracy. Altogether, Triage's auxiliary structures (TU, replacement state, LUT) require 18.875KB.

2) Overall Operation: On an L2 miss or prefetch hit to address A, Triage finds A's trigger in its TU and stores the new correlation in its metadata. Triage then issues metadata read requests and prefetches until the degree of four is met. Triage resizes its metadata partition every 50K accesses.

B. Triangel

Triangel improves upon the prefetch accuracy and storage efficiency of Triage with three main changes: (1) a confidence mechanism, which tracks the likelihood that a PC will produce accurate address correlations, (2) a metadata reuse buffer (MRB), which reduces the pressure on the LLC, and (3) a dynamic partitioning scheme based on set dueling, which maximizes the hit rate of both triggers and data. Figure 2 summarizes Triangel's overall organization.

1) Hardware Design: Triangel's metadata entries consist of a hashed trigger address (10 bits), an uncompressed prefetch target (31 bits), and a confidence bit, so 12 entries fit in an LLC block. To enable a prefetching lookahead of two, Triangel modifies the TU entries to track the last two addresses.

Triangel's TU entries have 3 confidence counters (12 bits), a sampling rate (4 bits), and a timestamp (32 bits). For a given PC, reuse confidence indicates whether correlations are used before eviction, and pattern confidence measures prefetch accuracy by tracking whether correlations repeat. Triangel tracks confidence by implementing two structures: (1) a history sampler (HS), which tracks reuse and pattern confidence, and (2) a second chance sampler (SCS), which provides leeway for reordering. Overall, Triangel requires 17.6KB per core.

2) Overall Operation: On an L2 miss or prefetch hit to address A by PC X, Triangel queries its TU for A's trigger address. Depending on X's lookahead bit, Triangel correlates A with either the last address or the one prior (1). Triangel's confidence counters determine whether to store X's correlations and set X's prefetch degree. If X can store correlations, Triangel checks its MRB to prevent redundant stores before writing to its metadata (2). If X has non-zero degree, Triangel searches for A first in its MRB and then in the metadata store, updating the MRB if an entry is found (3). If A's entry is found, Triangel issues a prefetch (4) and repeats step (3) and step (4) until X's prefetch degree is met.

By sampling correlations into its HS, Triangel measures X's (1) reuse confidence (i.e., whether X's correlations get reused prior to eviction) and (2) pattern confidence (i.e., whether X consistently produces the same correlations) (5). The HS evicts correlations into the SCS, which provides leeway in case the prefetch target doesn't occur immediately after its trigger (6).

Triangel's set-dueling partitioner tracks the number of trigger and data hits and resizes to best maximize both 7.

C. Issues with On-Chip Temporal Prefetchers

We now identify three issues with these on-chip prefetchers.

1) Pairwise Representation: Both Triage and Triangel store their metadata using a pairwise representation where each metadata entry consists of a trigger address and a prefetch target (see Figure 1a). For example, an access stream [A, B, C, D, E] is represented as pairs of correlated addresses: (A, B), (B, C), (C, D), (D, E). We see that addresses B, C, and D are stored twice in the metadata store. Furthermore, since their metadata entries are scattered throughout the metadata partition, higher degree prefetching requires multiple independent reads to the LLC, significantly increasing traffic.

2) Issues with Dynamic Partitioning: Dynamic partitioning is important because the utility of data and metadata can vary over time. Unfortunately, dynamic partitioning in Triangel incurs heavy data movement because the location for each entry is determined by a two-level metadata index function based on the trigger address: The first index function selects the LLC set, and the second selects a way within the selected set. Thus, each resize of the metadata store changes the number of ways allocated for metadata and changes the index function that maps metadata entries to LLC ways (see Figure 5a). Consequently, a change in indexing can leave metadata in the wrong way. Triangel's solution is to relocate the misplaced metadata after each resize, but each rearrangement operation could require shuffling up to 1MB of metadata in the LLC.

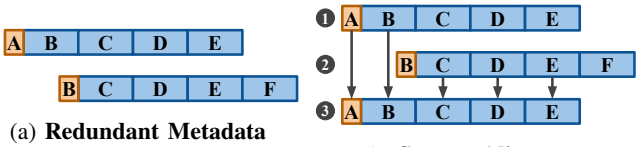
3) Trigger-Based Metadata Management: Both Triage and Triangel manage metadata as if they were managing cache data. For metadata replacement, they use cache replacement policies—Hawkeye [21] for Triage and SRRIP [4] for Triangel—that were designed to minimize data misses; when applied to temporal metadata, these policies minimize trigger misses without regard to the utility of the metadata. Similarly, for dynamic partitioning, Triage uses Hawkeye to find the size that maximizes trigger hit rate, and Triangel uses set dueling [42] to optimize the combined hit rate of data and trigger accesses, all without regard to the utility of the metadata.

IV. Our Solution

Our solution builds on the observation that temporal metadata contains semantic information that regular data does not. Thus, our Streamline prefetcher uses a stream-based representation that is particularly amenable to the constraints of on-chip temporal prefetching, and our prefetcher considers the prefetch utility of each entry when managing its metadata. The result is a simplified design and an on-chip temporal prefetcher that maximizes storage efficiency.

A. Stream-Based Representation

Streamline's stream-based representation eliminates the inherent redundancy in the pairwise representation used by prior on-chip temporal prefetchers. For example, if we utilize stream-based metadata entries of length four (i.e., those that store four correlations per metadata entry), the space saved from eliminating this pairwise redundancy enables each cache block to hold 4 additional correlations. Since a pairwise representation can hold 12 correlations per cache block, a stream-based metadata store can hold 33% more correlations than a pairwise metadata store.



(a) **Redundant Metadata** (b) **Stream Alignment**

Fig. 3: **Misalignment Causes Redundant Metadata: (a)** *Misalignment* occurs when an old entry and a new entry overlap (e.g., [B,C,D,E]) but have different triggers. **(b)** By combining the old entry 1 and the new entry 2 into an aligned entry 3, stream alignment eliminates this redundancy.

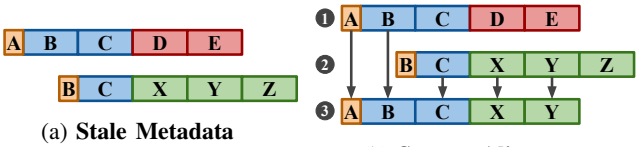
Moreover, our stream-based representation provides several additional benefits. First, since every address in a stream entry was accessed by the same load PC, stream-based entries are naturally PC-localized, which yields higher prefetch accuracy. By contrast, pairwise entries could have been inserted by any PC, so higher degree prefetches can jump among streams. Second, streams reduce metadata traffic by a factor of their length. For example, Streamline's stream length of four can reduce traffic by up to 4×. Finally, the use of our stream-based representation eliminates the high cost of metadata store resizing without sacrificing any prefetch coverage.

B. Resolving Stream-Based Problems

While a stream-based format has many valuable properties, it also presents significant new challenges: (1) Because there are fewer triggers, the likelihood of missing a trigger increases and the prefetch coverage decreases; (2) because it can store overlapping entries for the same stream, stream misalignment creates staleness and a new form of redundancy; and (3) stream entries are larger, so a naive implementation lowers associativity, which can lead to increased conflict misses in the metadata store. The remainder of this subsection presents our solutions to these issues.

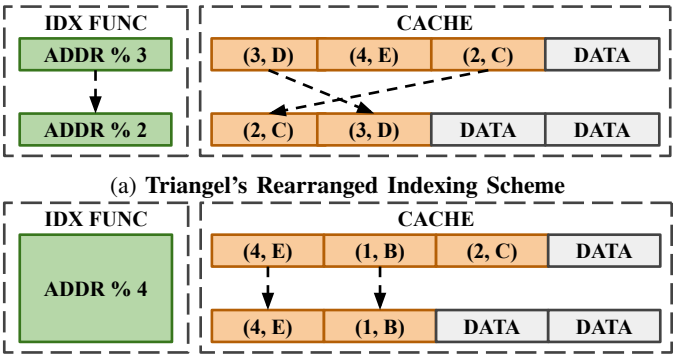
1) Missed Triggers: We empirically find (see Section V-C1) that stream lengths of five or more lose significant prefetch coverage from missed triggers. We also find that a stream length of four sits at the inflection point that optimizes both the storage capacity and the number of missed triggers. We therefore design Streamline with a stream length of four.

2) Stream Misalignment: Figure 3a shows an old entry and a new entry that share an overlapping substream ([B,C,D,E]) but have different triggers. These two entries are *misaligned* because naively inserting the new entry into the metadata store would create redundancy with the old entry. Moreover, Figure 4a shows that the old entries can hold stale metadata, resulting in lost prefetch coverage ([X,Y]) and useless prefetches ([D,E]). In Figures 3b and 4b, we eliminate



(a) **Stale Metadata** (b) **Stream Alignment**

Fig. 4: **Misalignment Produces Stale Metadata: (a)** Misalignment can result in **stale metadata**; here, the stream entry $[A, B, C, D, E]$ should be updated to $[A, B, C, X, Y]$, but because the stream entries are misaligned, the trigger A will continue to prefetch $[D, E]$ (useless prefetches) instead of prefetching $[X, Y]$ (lost coverage). **(b)** By overwriting old entries 1 with **updated metadata** from new entries 2, stream alignment eliminates this stale metadata.



(a) **Triangel's Rearranged Indexing Scheme**

(b) **Streamline's Filtered Indexing Scheme**

Fig. 5: **Partition Indexing**: When Triangel resizes from 3 **metadata ways** to 2, the **index function** changes, creating *misplaced metadata* that need to be rearranged. By contrast, Streamline's metadata store uses a static index function, preventing metadata from being misplaced.

misalignment by performing *stream alignment*, which creates an aligned entry 3 from the trigger of the old entry 1 and the updated correlations of the new entry 2. The new entry's leftover correlations are then used to bootstrap the next entry.

The matching old entry for a new entry can be found in Streamline's per-PC metadata buffer, which holds recently fetched entries for the current stream. Thus, for each completed stream entry E, Streamline performs stream alignment by searching its metadata buffer for any entry that contains E's trigger address. (It skips any entry that has E's trigger address as its final address because there is no overlap.) In Section V-C2 we show that a 3-entry per-PC metadata buffer is sufficient for finding the majority of these misaligned entries.

3) Increased Conflict Misses: Stream-based metadata entries are larger than pairwise entries, so fewer entries can fit in each metadata way (4 vs 12). Thus, if we were not careful, our stream-based representation would lower associativity and increase the number of conflict misses in the metadata store.

We resolve this issue by replacing both Triangel's way-partitioning scheme and its complicated two-level index function with a simpler unified *tagged set-partitioning scheme*. Rather than partitioning by ways, our tagged set-partitioning scheme instead partitions by LLC sets. Each allocated set dedicates a fixed number of ways to metadata storage: 8 (i.e., half of them). Moreover, rather than store the metadata trigger tag with the rest of the entry within an allocated metadata way, our scheme instead stores part of these trigger tags within the existing LLC tag store, which simplifies the design by eliminating the need for a secondary index function.

Thus, under our tagged set-partitioning scheme, each metadata set has a fixed number of allocated ways, and metadata entries can be freely[1] placed in any of those ways, creating an effective associativity of 32 (4 entries in each of the 8 allocated ways), thereby eliminating any issues from low associativity. Our scheme is made possible by our stream-based representation, which reduces the number of metadata entries and, consequently, the number of tags. By contrast, a pairwise representation cannot utilize the LLC tag store for this purpose, because it would require 120 tag bits per way, which far exceeds the size of a cache tag.

[1]Because partial trigger tags can alias, entries that have the same partial tag are stored in the same LLC way so Streamline would need only 1 LLC read per access. In Section V-D5 we show that partial tag aliasing occurs infrequently and minimally impacts performance.

C. Eliminating Expensive Repartitioning

Figure 5a shows that when Triangel resizes the metadata partition, the allocation of ways between data and metadata changes, as does the metadata index function, resulting in *misplaced metadata entries,* which are located in the wrong way. Triangel handles this issue by rearranging all of the misplaced metadata after a repartition. Unfortunately, this scheme is extremely costly because it can require shuffling up to 1MB of metadata in the LLC.

To solve this problem, we introduce *filtered indexing,* which uses a fixed index function that matches that of the maximum partition size; any metadata entry that is mapped to a way (or set) allocated for data is filtered out and discarded. For example, in the top of Figure 5b where there are only 3 ways allocated, an entry $(3, V)$ would get filtered out because it would have mapped to a way allocated for data. In other words, filtered indexing avoids the creation of misplaced metadata.

It might appear that filtering would create gaps in the recorded stream and reduce prefetch coverage, but by storing our metadata as streams, we can cover these gaps by realigning the stream to select a different trigger. For example, consider (1) the access stream $[A_1, B, A_2, A_3]$, where B is filtered and the A_i accesses are not and (2) the current stream entry (B, A_2, A_3). Since B is a filtered trigger, that entry would be discarded; however, we can instead realign this entry to the prior access A_1, creating a realigned entry (A_1, B, A_2) that is not filtered. With realignment, we can cover many triggers that filtered indexing would otherwise discard. By contrast, a pairwise format cannot be realigned because it has no flexibility to change the trigger address—any filtered entries would simply result in lost prefetch coverage. In

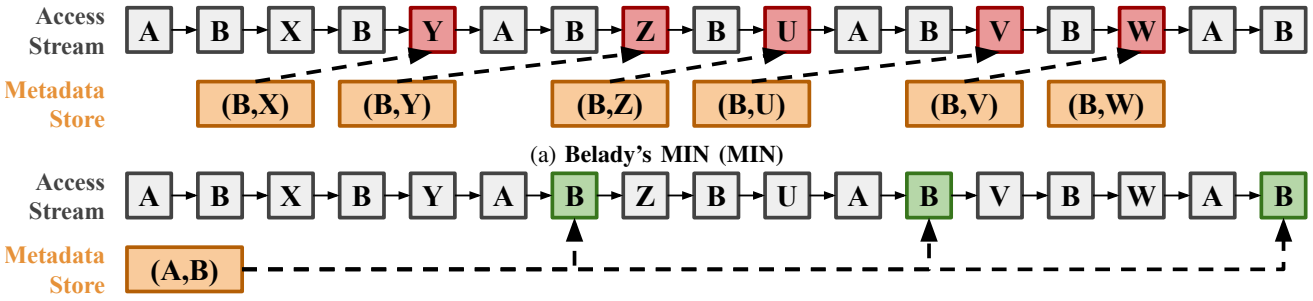


Fig. 6: **Utility-Aware Replacement**: An illustration that TP-MIN better caches temporal metadata than MIN. We represent the access stream as a sequence of squares and the contents of a 1-entry metadata store as orange rectangles. **(a)** Since MIN maximizes trigger hits, it stores correlations with the trigger B. But because B's target address is unstable, MIN **fails to cover** any access. **(b)** TP-MIN maximizes correlation hits, so it **successfully covers** 3 accesses by storing the correlation (A, B).

TABLE I: **Partitioning**: Across all schemes, only our filtered tagged set-partitioning (FTS) reduces conflict misses and fixes costly repartitioning. All other combinations of Rearranged (R) / Filtered (F) indexing, Untagged (U) / Tagged (T), and Way (W) / Set (S) partitioning don't address both issues.

Partition Scheme	Low Associativity		Expensive Repartitioning
	Small Partition	*Big Partition*	
RUW	✗	✗	✗
FUW	✗	✗	✓
RUS	✗	✗	✗
FUS	✗	✗	✓
RTW	✗	✓	✓
FTW	✗	✓	✓
RTS	✓	✓	✗
FTS (ours)	✓	✓	✓

Section V-D6 we show that *stream realignment* recoups most of the performance that would have been lost to filtering.

Table I shows that only Streamline's FTS partitioning scheme addresses the low associativity across partition sizes and eliminates the need for expensive repartitioning.

D. Utility-Aware Metadata Management

Whereas Triangel utilizes prefetch utility only to filter out inaccurate metadata entries, Streamline utilizes it in every aspect of its metadata management. Specifically, we approximate prefetch utility by considering the entire correlation as a unit instead of considering the trigger address alone. In doing so, Streamline better controls the contents and size of its metadata store and thus achieves better storage efficiency.

1) Temporal-Prefetching-MIN: Triangel [4] uses the SRRIP [22] policy for metadata entry replacement because its authors found minimal benefit from the use of Hawkeye [21]. However, Hawkeye learns from Belady's optimal replacement policy [9], and we observe that Belady's MIN is not optimal for the replacement of temporal prefetching metadata.

Prior work [54] applied Belady's MIN to temporal metadata by maximizing the number of trigger address hits, but this approach is suboptimal because it doesn't consider the utility

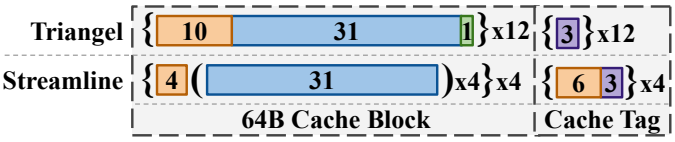


Fig. 7: **Metadata Format**: Triangel stores 12 correlations (trigger, target, conf) per cache block, whereas Streamline stores 16 correlations across four stream entries. Triangel stores replacement state in the tag array. Streamline additionally stores partial trigger tags.

of the prefetch target associated with those triggers. Figure 6a shows an example where maximizing the trigger hit rate can generate many useless prefetches because the trigger address doesn't consistently map to one prefetch target.

To address this issue, we introduce Temporal-Prefetching-MIN, or TP-MIN, which maximizes the hit rate of the entire correlation rather than just its trigger address. Whereas Belady's MIN evicts the correlation whose trigger is used furthest in the future, TP-MIN instead evicts the correlation used furthest in the future. Figure 6b shows that TP-MIN can enable more accurate prefetching than MIN.

2) Utility-Aware Dynamic Partitioning: Triangel sizes its metadata store with a set dueling scheme [42] that maximizes the hit rate of both data and triggers. But as Section IV-D1 explains, the trigger hit rate does not directly correlate with prefetching performance, so we instead modify this scheme to maximize the hit rate of data and accurate correlations. Whereas Triangel weights each metadata hit equally, we score correlation hits based on the current global prefetch accuracy, which we show in Section V-D3 better sizes Streamline's metadata store. Section IV-E4 provides further details.

E. Hardware Design

We now describe Streamline's hardware design, highlighting its differences from prior work. We then detail its operations, as well as the overhead of our auxiliary structures.

1) Stream-Based Metadata: Figure 7 shows that Streamline's metadata entries consist of a 10-bit hashed trigger and

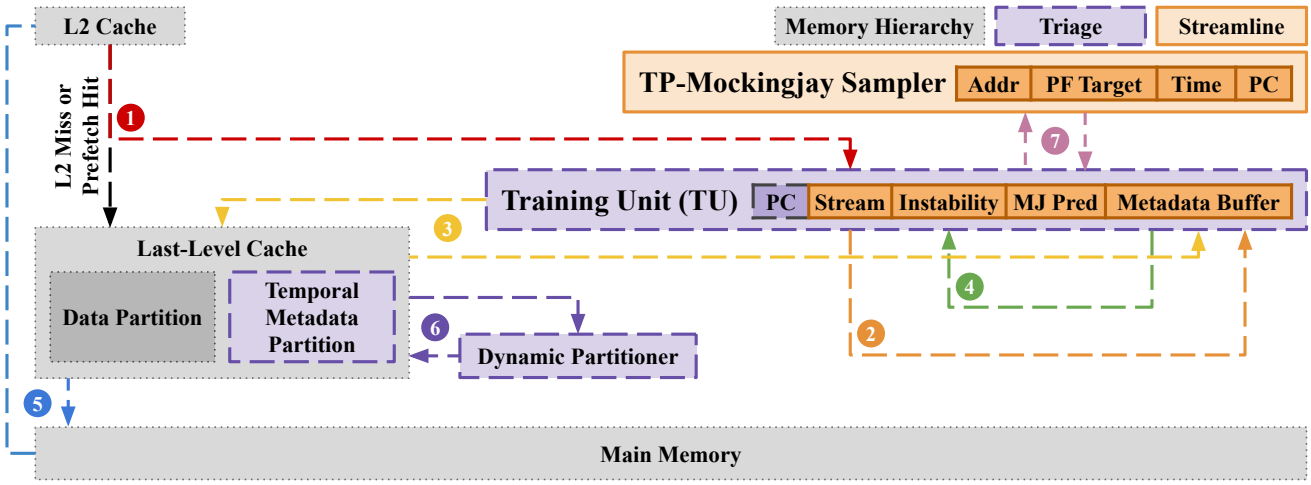


Fig. 8: **Overview of Streamline**

four 31-bit prefetch targets. To fit four stream entries into a cache block, we spill trigger bits into the LLC tag store (see Section IV-B3). To locate its metadata entries within the metadata store, Streamline performs a standard tag check.

2) Training Unit: Like prior work, Streamline's training unit records prior accesses for each PC but tracks the last stream instead of the last address. Whereas Triangel has a shared Metadata Reuse Buffer, we instead augment each of Streamline's TU entries with a small 3-entry stream metadata buffer. A shared metadata buffer like Triangel's would be more storage efficient, but Streamline needs to have metadata entries for each PC available so that it can perform stream alignment. We evaluate different stream metadata buffer sizes in Section V-C2 and empirically determine that a size of three offers a good balance of storage cost and performance.

3) Set-Partitioning: Streamline creates its metadata store by taking a subset of LLC sets and allocating 8 ways in each of these sets for metadata. Given a 2MB LLC, Streamline creates a 1MB store by allocating every set for metadata, a 0.5MB store by allocating every other set for metadata, a 0.25MB store by allocating every 4^{th} set for metadata, and so on.

4) Dynamic Partitioning: Whereas Triangel scores all metadata hits equally, Streamline scores them based on prefetch accuracy. Moreover, whereas Triangel has 9 partition sizes (0 to 8 ways), Streamline has 3: 0MB, 0.5MB, and 1MB. Each data hit increments the set dueling counters by 16. Each metadata hit increments based on prefetch accuracy: for 10% to 25% prefetch accuracy, increment by 2, for 25% to 50% increment by 3, for 50% to 70% increment by 4, for 70% to 90% increment by 6, for 90% to 95% increment by 7, and for 95+% increment by 8. Streamline tracks prefetch accuracy in epochs of 2048 prefetches. Since a 0MB partition cannot issue prefetches, we permanently allocate and sample 64 metadata sets. Streamline resizes every 2^{15} sampled accesses.

5) Metadata Replacement: Streamline's metadata replacement policy is a modified version of Mockingjay [45]—which we refer to as TP-Mockingjay—that emulates TP-MIN instead of Belady's MIN. Details about Mockingjay can be found in the original paper; we now focus on our modifications. TP-Mockingjay samples 8 LLC sets, and for each sampled set it manages a 32-set, 10-way sampler. To learn from TP-MIN, we modify the sampler entries to store correlations. To reduce storage overhead, we learn the reuse distance for just the first correlation in a stream entry. Whereas Mockingjay uses 5-bit ETRs, we find that 3 bits suffice because temporal metadata has more consistent reuse than raw data.

6) Stability-Based Degree Control: For each PC, Streamline sets the prefetch degree with a simple metric that tracks how often a metadata entry is inserted into the metadata buffer. In other words, this metric measures how often an attempted prefetch requires a metadata read, which is analogous to measuring the stability of a PC's access stream.

For a stream length of four, a stable PC should hit in the metadata buffer 75% of the time and only need to fetch a new entry every four accesses. By contrast, an unstable PC would fetch entries more frequently because the addresses it accesses miss in the metadata buffer. Thus, Streamline tracks PC instability in epochs of 1024 accesses and prefetches at degree four whenever there are less than 400 insertions, degree three whenever there are less than 600, degree two whenever there are less than 800, and prefetches at degree one otherwise.

7) Overall Operation: Figure 8 shows the details of Streamline's training and prefetch operations.

Training: On an L2 miss or prefetch hit to address A by PC X, Streamline appends A to X's stream entry E (1). If E is full, the metadata buffer is checked for E's trigger address. If found, Streamline performs stream alignment on E and the matching entry (2). Either the unaligned E or the resulting aligned entry is then written back to the metadata partition.

Prefetching: If A misses in the metadata buffer, Streamline sends a read request for A to the metadata store and updates the metadata buffer (3). If a new entry was read, then X's instability counters are updated. Every 1024 accesses, Streamline sets X's degree based on its instability (4). If A now hits in the metadata buffer, Streamline issues a prefetch (5) and

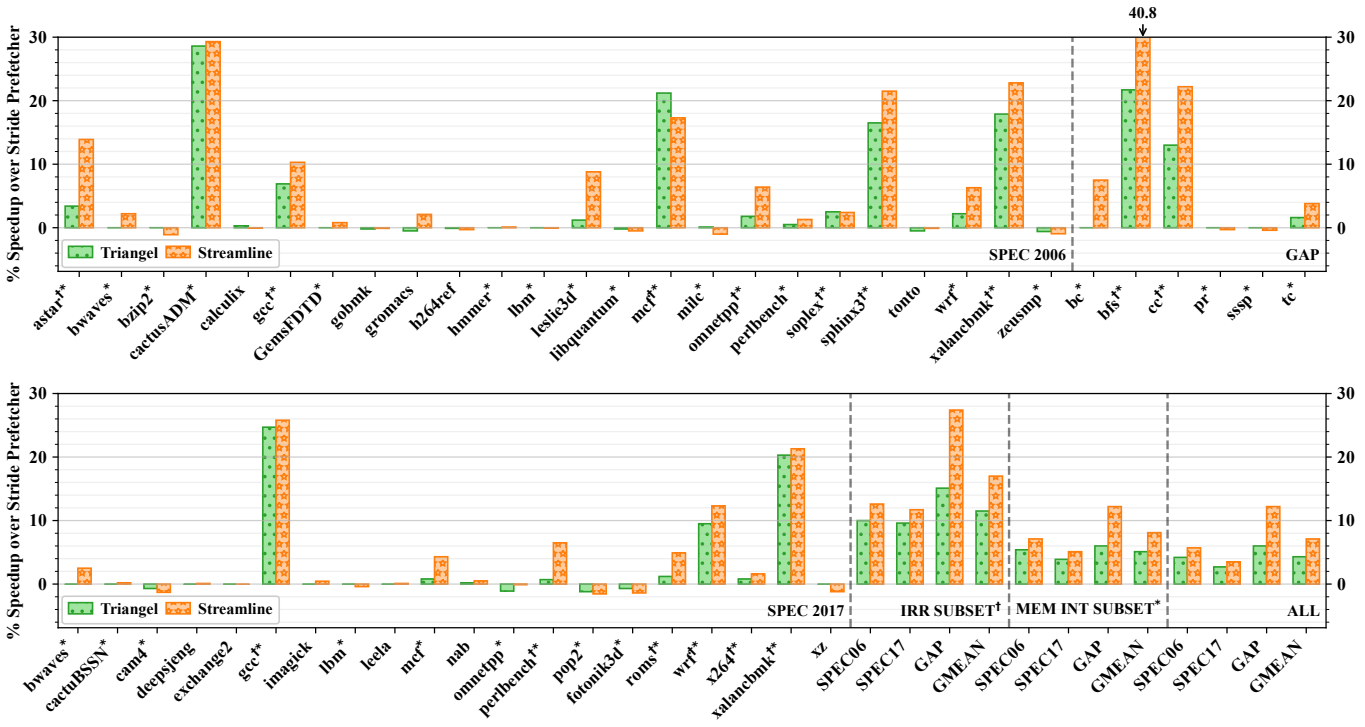


Fig. 9: **Single-Core**: Streamline outperforms Triangel across the irregular subset, memory-intensive subset, and all benchmarks.

TABLE II: **Simulator System Parameters**

Core	4GHz, 6-wide OoO, 352-entry ROB, hashed-perceptron [25]
L1I	32KB, 8-way, 4-cycle latency, 8 MSHRs, 2 R/W ports
L1D	48KB, 12-way, 5-cycle latency, 16 MSHRs, 2 R/W ports PC-Localized Stride Prefetcher (degree = 3)
L2	512KB, 8-way, 10-cycle latency, 32 MSHRs, 1 R/W port
LLC	2MB/core, 16-way, 20-cycle latency, 64 MSHRs, 1 R/W port
DRAM	3200 MT/S, 8B channel width, 64K rows tCAS = tRP = tRCD = 12.5, 8 banks/rank 1/2/4/8C: 1/2/2/4 channels, 1/1/2/2 ranks/channel

repeats steps 3 to 5 until X's prefetch degree is met or until Streamline misses in both the metadata buffer and store.

To resize the metadata partition every epoch, Streamline's utility-aware partitioner is updated—on an L2 miss, a prefetch hit, or a metadata access—similar to Triangel's set-dueling partitioner 6. To update its reuse distance predictor, TP-Mockingjay samples completed stream entries 7.

8) Storage Overhead: Each training unit (TU) entry stores a 3-entry metadata buffer, the current stream, two 10-bit instability counters, and TP-Mockingjay's 3-bit prediction. Thus, a 256-entry TU requires 17.8KB. For TP-Mockingjay, each sampler entry stores a valid bit, two 8-bit address hashes for the trigger and prefetch target, an 8-bit timestamp, and an 8-bit hashed PC, for a total of 10.3KB. TP-Mockingjay also requires a 5-bit per-set clock for another 1.3KB. Overall, Streamline's auxiliary structures require 29.4KB per core. Since increasing Triangel's size to 30KB doesn't affect performance, we faithfully model the original size.

V. Evaluation

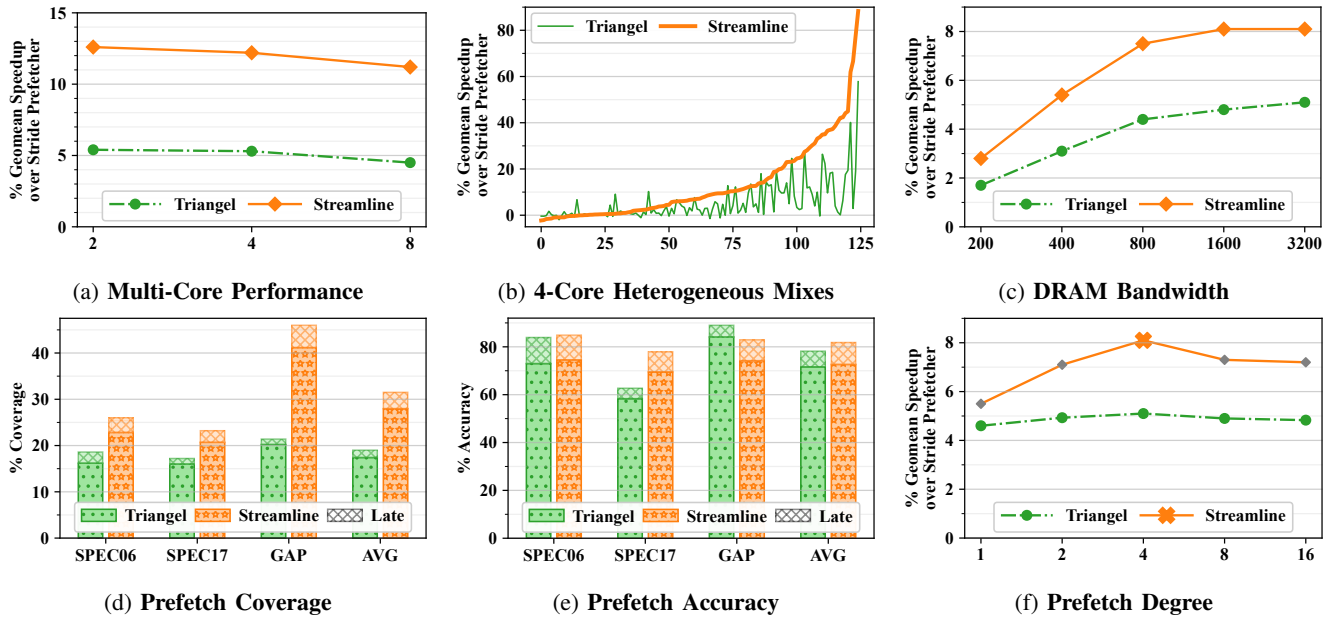
This section presents our empirical evaluation of Streamline, whose benefit over Triangel is best seen in a multi-core setting. But before presenting these results in Section V-B2, we first describe our methodology and present single-core results.

A. Methodology

1) Simulator: We evaluate Streamline using ChampSim [18], a trace-driven simulator. We model an Intel Ice Lake core [3]. Details are provided in Table II. In our single-core evaluation, we warmup for 200 million instructions and evaluate for 800 million instructions. In our multi-core evaluation, we scale up the LLC and DRAM parameters, warmup for 200 million instructions, and evaluate for 200 million instructions.

2) Baseline: We compare Streamline to the state-of-the-art temporal prefetcher, Triangel [4]. Both prefetchers train on L2 misses and on prefetch hits, prefetch into the L2 at a max degree of four, and store metadata in the LLC. We model the traffic, latency, and port contention of metadata accesses. Our baseline system has an L1D IP-Stride prefetcher, and in Section V-B6 we also evaluate with Berti [35], the state-of-the-art L1D prefetcher. And in Section V-B7 we evaluate with L2 stride prefetchers (IPCP [37], Bingo [7], and SPP-PPF [14]).

3) Benchmarks: We evaluate on all memory-intensive benchmarks (>1 LLC MPKI) in the SPEC 2006 [1], SPEC 2017 [2], and GAP [8] suites. For our multi-core evaluation, we simulate 150 mixes of the memory-intensive workloads



(a) **Multi-Core Performance** (b) **4-Core Heterogeneous Mixes** (c) **DRAM Bandwidth**

(d) **Prefetch Coverage** (e) **Prefetch Accuracy** (f) **Prefetch Degree**

Fig. 10: **Performance Analysis**: **(a)** Streamline outperforms Triangel across core counts, **(b)** winning on 77% of our mixes. **(c)** Streamline outperforms Triangel across bandwidth levels. **(d)** Streamline's high storage efficiency significantly improves prefetch coverage, **(e)** at no loss in prefetch accuracy. **(f)** Since streams provide more context, Streamline can profitably increase degree up to its stream length.

for each of our evaluated core counts: 2, 4, and 8. Similar to prior evaluations of Triage [54] and Triangel [4], Figures 9 and 11a additionally show results for an irregular subset of benchmarks. As in prior work, we create this subset by taking all benchmarks with at least 5% headroom under an idealized Triage prefetcher that is given unlimited metadata storage.

B. Performance Analysis

1) Single-Core Results: Figure 9 shows that Streamline (8.1%) outperforms Triangel (5.1%) across memory-intensive benchmarks over our baseline with a strided L1 prefetcher. Specifically, Streamline outperforms Triangel by 2.7, 1.2, and 6.2 percentage points on SPEC 2006, SPEC 2017, and GAP, respectively. Triangel outperforms Streamline on SPEC 2006 mcf because it bypasses metadata entries from the load PCs that perform scans (i.e., data accesses with no temporal reuse). Since Streamline does not have a bypassing mechanism, it must insert these non-temporal entries and evict more valuable entries. Moreover, because Streamline permanently allocates 64 sets to metadata, it suffers a larger performance loss than Triangel on workloads with little irregularity, such as bzip2.

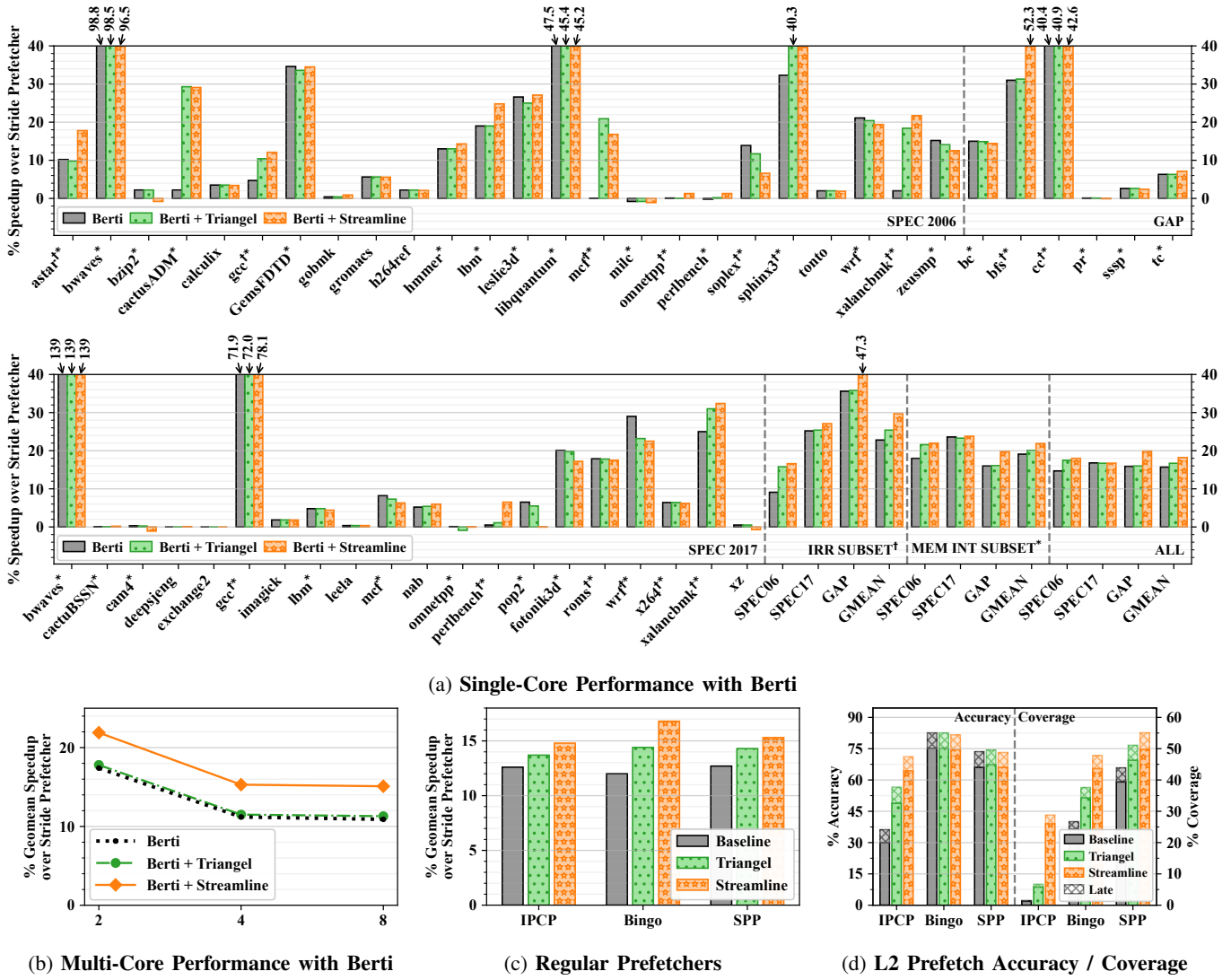
On the irregular subset, Streamline (17%) likewise outperforms Triangel (11.5%). Specifically, Streamline outperforms Triangel by 2.6, 2.1, and 12.3 percentage points on SPEC 2006, SPEC 2017, and GAP, respectively.

2) Multi-Core Results: Figure 10a shows that Streamline outperforms Triangel by wider margins on multi-core systems, showing that Streamline better handles diverse workloads than Triangel. Specifically, Streamline sees wins of 7.2, 6.9, and 6.7 percentage points for 2, 4, and 8 cores, respectively. Moreover, Figure 10b shows that Streamline outperforms Triangel in 77% of the 4-core heterogeneous mixes.

3) DRAM Bandwidth: Figure 10c shows that in bandwidth-limited settings where off-chip traffic is precious, Streamline enjoys a performance win because its storage efficiency increases both prefetch coverage and accuracy with a minimal increase to off-chip traffic. Specifically, Streamline maintains a 3 to 3.3 percentage point margin for moderate bandwidth levels and a 1.1 to 2.7 percentage point margin at lower levels. Overall, Streamline only increases off-chip traffic by 1.3 percentage points over Triangel (see Figure 14).

4) Prefetch Coverage and Accuracy: Figure 10 shows that Streamline has better prefetch coverage (+12.5 percentage points) and prefetch accuracy (+3.6 percentage points) than Triangel. Specifically, Streamline sees 7.4, 5, and 24.6 percentage points better prefetch coverage across SPEC 2006, SPEC 2017, and GAP, respectively. Notably, our ablation study in Section V-D4 shows that naively replacing a temporal prefetcher's pairwise format with a stream-based one enables it to achieve significantly better prefetch coverage (+7.6 percentage points) than Triangel.

While Triangel's metadata filtering improves prefetch accuracy, it does so at the cost of prefetch coverage. By contrast, Streamline's TP-Mockingjay replacement policy and stream-based metadata format enable it to have better prefetch accuracy at no loss in prefetch coverage. In particular, TP-Mockingjay prioritizes the eviction of entries likely to issue useless prefetches. Moreover, stream-based entries provide greater context than pair-based entries can. That is, when Streamline hits in its metadata buffer, Streamline issues a



(a) **Single-Core Performance with Berti**

(b) **Multi-Core Performance with Berti**

(c) **Regular Prefetchers**

(d) **L2 Prefetch Accuracy / Coverage**

Fig. 11: **Regular Prefetchers**: **(a)** With Berti in the L1D, Streamline still outperforms Triangel across the irregular subset, the memory-intensive subset, and all benchmarks. **(b)** Streamline significantly widens the gap on multi-core systems. **(c)** With L2 stride prefetchers, Streamline sees similar wins because **(d)** it provides twice as much additional prefetch coverage as Triangel.

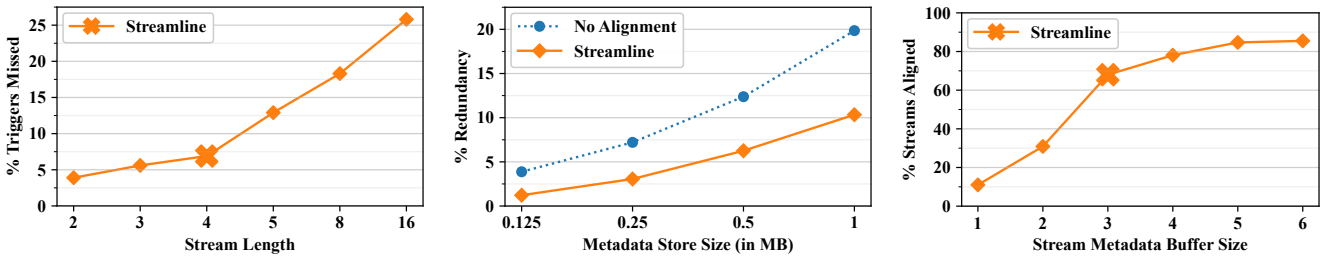
prefetch that has the historical context of (1) the current trigger access and (2) the previous trigger access that brought that entry into the metadata buffer. In other words, Streamline's metadata buffer hits provide context analogous to temporal prefetchers that utilize longer trigger histories [6], [16], [48].

5) Prefetch Degree: Figure 10f shows that while Triangel is insensitive to the degree, Streamline performs best at degree four—its stream length—because of our metadata format. For pairwise formats, higher degree prefetching can jump among different streams, which reduces prefetch accuracy. By contrast, our stream-based format only stores in each entry accesses from a single stream, enabling more accurate prefetching at higher degrees up to the stream length.

6) Upper-Level Prefetchers: To evaluate Streamline and Triangel in the presence of an aggressive stride prefetcher, we use a baseline with Berti in the L1D. Figure 11b shows that in a multi-core setting, Triangel provides no performance benefit, while Streamline outperforms Triangel by 4.1, 3.9, and 3.8 percentage points for 2, 4, and 8 cores, respectively.

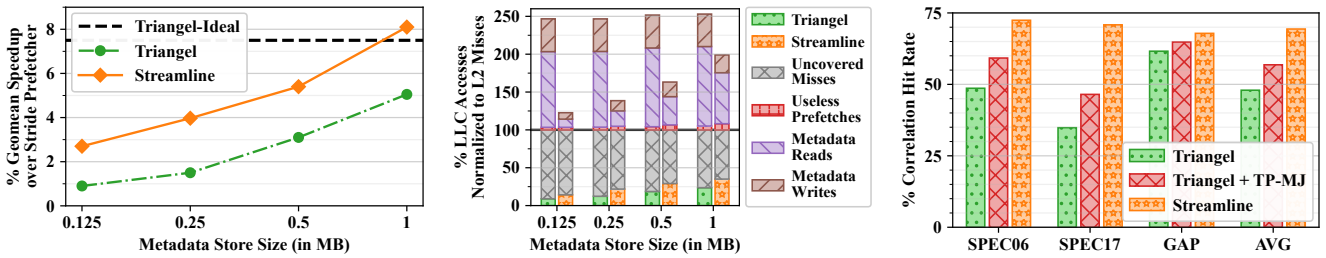
Figure 11a shows that on one core Streamline (22%) outperforms both Triangel (20.1%) and Berti (19.1%). Specifically, Streamline outperforms Triangel by 0.5, 0.8, and 3.7 percentage points and Berti by 4, 0.5, and 3.8 percentage points on SPEC 2006, SPEC 2017, and GAP, respectively. On the irregular subset, Streamline further outperforms Triangel (+3.5 percentage points) and Berti (+6.1 percentage points).

7) L2 Stride Prefetchers: To evaluate Streamline and Triangel in the presence of regular L2 prefetchers, we use baselines with IPCP [37], Bingo [7], and SPP-PPF [14]. Figure 11c shows that Streamline outperforms Triangel by 1.1, 2.4, and 1 percentage points and the baselines by 2.2, 4.8, and 2.6 percentage points for IPCP, Bingo, and SPP-PPF, respectively.



(a) **Missed Triggers** (b) **Redundancy** (c) **Stream Alignment Rate**

Fig. 12: **Fixing Stream Issues**: **(a)** A stream length of four maximizes performance by optimizing both the rate of missed triggers and the storage capacity. **(b)** Stream alignment halves metadata redundancy. **(c)** A 3-entry metadata buffer sits at the inflection point that balances the stream alignment rate and storage overhead.



(a) **Storage Efficiency** (b) **Metadata Traffic** (c) **Correlation Hit Rate**

Fig. 13: **Efficient Metadata Management**: **(a)** Streamline matches Triangel's performance even when Streamline is given a metadata store that is half as large as Triangel's. **(b)** Streamline significantly reduces metadata traffic through its use of stream-based metadata and filtered indexing. **(c)** Streamline significantly improves correlation hit rate by considering prefetch utility in its replacement decisions.

Figure 11d shows that Streamline's wins come from providing twice as much additional prefetch coverage as Triangel.

C. Resolving Stream-Based Problems

We now present results that demonstrate how Streamline handles the unique problems that arise from the use of streams.

1) Missed Triggers: Figure 12a shows that a stream length of four performs best by maximizing storage capacity while minimizing the loss in prefetch coverage from having one trigger per stream. In particular, lengths of four, eight, and sixteen hold 16 correlations per way, whereas lengths of two, three, and five can hold only 14, 15, and 15 correlations, respectively. Moreover, the rate of missed triggers is stable up to a length of four, which misses 6.8% of triggers relative to a pairwise format, and then rapidly increases to 25.8%. Both the storage capacity and rate of missed triggers impact prefetch coverage, which peaks at 31.5% for a length of four and decreases to 24.1% for a length of sixteen and 28.6% for a length of two. Thus, while lengths of eight and sixteen also maximize storage capacity and have 1.9 and 3.2 percentage points more prefetch accuracy than a length of four, their high rate of missed triggers ultimately degrades performance.

2) Redundancy: Figure 12b shows the impact of stream alignment on metadata redundancy. We measure redundancy as a function of metadata store size, but we exclude benign forms of redundancy that disambiguate streams. To illustrate *benign redundancy,* consider the entries (C, A, T) and (D, A, Y). Even though address A appears redundantly, Streamline uses it in a greater context—(C, A) and (D, A)—to determine when to prefetch T or Y. Because a pairwise format cannot distinguish these contexts, it would oscillate between storing (A, T) and (A, Y) and mispredict. We find that 31% of Streamline's redundancy is benign and that stream alignment halves the rate of redundancy.

Figure 12c shows the stream alignment rate: We measure the rate at which Streamline detects that a new metadata entry is redundant with one in the cache; we see that Streamline needs a metadata buffer with at least 3 entries to effectively align streams. With a 1-entry buffer, Streamline aligns only 11% of the redundant entries, but with a 3-entry buffer it aligns 67% of them, improving prefetch coverage by 1.2 percentage points over a 1-entry buffer. Larger buffers are not helpful: They improve the stream alignment rate but do not further improve prefetch coverage; instead, they only increase overhead.

3) Conflict Misses: Tagged set-partitioning simplifies our design by eliminating Triangel's two-level index function. Moreover, by enabling full-associativity, it improves Streamline's correlation hit rate by 10.8 percentage points and improves performance by an additional 2.2 percentage points over our baseline.

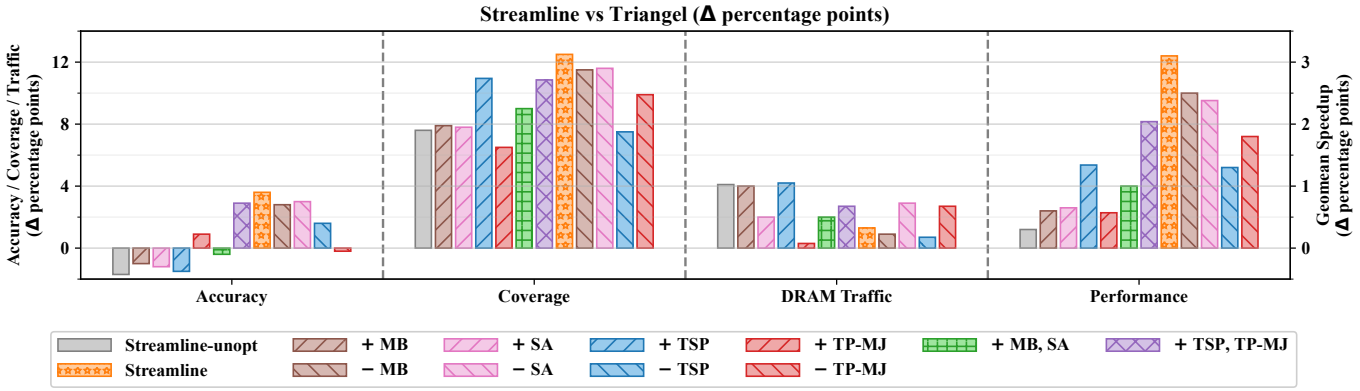


Fig. 14: **Ablation Study**: Streamline's component structures improve performance by significantly improving storage efficiency and thus increasing prefetch coverage (without decreasing prefetch accuracy). We show the impact of **(1)** adding (+) each structure on top of an unoptimized stream-based prefetcher (Streamline-unopt) and **(2)** removing (-) each structure from our complete prefetcher (Streamline). We see that neither a metadata buffer (+ MB) nor stream alignment (+ SA) alone affects prefetch coverage, but together they (+ MB, SA) increase prefetch coverage by reducing redundancy. Tagged set-partitioning (+ TSP) significantly improves prefetch coverage by reducing conflict misses, and TP-Mockingjay (+ TP-MJ) significantly improves prefetch accuracy by prioritizing high-utility metadata.

D. Understanding Our Benefits

We now present additional results that illustrate the source of Streamline's performance—its high storage efficiency.

1) Storage Efficiency: Figure 13 shows that Streamline is significantly more storage efficient than Triangel. In particular, we make two observations: (1) When given half of Triangel's metadata allocation, Streamline outperforms Triangel, and (2) when both Streamline and Triangel are given 1MB of metadata capacity, Streamline outperforms Triangel, even when Triangel has dedicated storage outside of the LLC (Triangel-Ideal).

2) Metadata Traffic: Figure 13b shows that our stream-based metadata and filtered indexing enable Streamline to significantly reduce metadata traffic. In particular, at 1MB, Streamline has 61% of Triangel's traffic. Since there is no filtering at 1MB, Streamline's traffic reduction comes purely from its stream-based metadata. Although a stream length of four should reduce traffic by $4\times$, its effectiveness is diminished because stream alignment writes back partial updates. The impact of filtered indexing increases as the size of the metadata store decreases. For example, at 0.125MB, filtered indexing reduces Streamline's traffic to 13% of Triangel's traffic.

3) Utility-Aware Metadata Management: To illustrate the benefits of optimizing for prefetch utility, we first compare a variant of Streamline that uses Belady's MIN with one that uses TP-MIN. By discarding entries that see no future use (i.e., entries that are unlikely to issue useful prefetches), Streamline-TP-MIN achieves a higher correlation hit rate (+9.3 percentage points) which improves prefetch accuracy (+4 percentage points) and speedup (+1.9 percentage points). Figure 13c shows that TP-Mockingjay enables Streamline to achieve a better correlation hit rate (+21.5 percentage points) than Triangel, and it also enables Triangel to close a third of the gap. We also compare Streamline against a variant that uses Triangel's partitioner, and we find that Streamline selects the best partition size more often (+22 percentage points).

4) Ablation Study: Figure 14 ablates various components of Streamline's design and shows how each contributes to its high prefetch coverage. In particular, we measure (1) the impact of adding each component on top of an unoptimized stream-based prefetcher (Streamline-unopt), which utilizes only our stream-based metadata format and (2) the impact of removing each component from our complete Streamline prefetcher.

We see that Streamline-unopt achieves better prefetch coverage (+7.6 percentage points) than Triangel. Thus, by eliminating the redundancy inherent to pairwise metadata formats, our stream-based metadata format alone significantly increases storage efficiency (and therefore prefetch coverage).

We observe that both a 3-entry metadata buffer (+ MB) and the stream alignment operation (+ SA) alone have marginal effects on prefetch coverage and performance. Without a sufficiently large metadata buffer, Streamline-unopt will often fail to perform stream alignment (see Section V-C2). Thus, these two optimizations need to be combined (+ MB, SA) to substantially reduce redundancy, which enable Streamline-unopt to achieve better prefetch coverage (+8.7 percentage points) and speedup (+1 percentage point) than Triangel.

We further observe that tagged set-partitioning (+ TSP) significantly improves prefetch coverage by reducing metadata conflict misses, and TP-Mockingjay (+ TP-MJ) significantly improves prefetch accuracy by prioritizing useful metadata entries. Similar to the prior two optimizations, the combination of these components is also synergistic because the increased cache associativity from tagged set-partitioning enables TP-Mockingjay to make better metadata replacement decisions. Thus, together they (+ TSP, TP-MJ) achieve better prefetch coverage (+10.7 percentage points), prefetch accuracy (+2.9 percentage points), and performance (+2.1 percentage points)

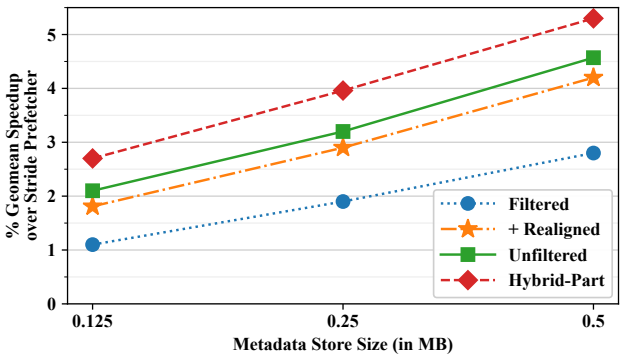


Fig. 15: **Mitigating Filtering Coverage Loss**: Realignment recoups 72% to 79% of the performance lost to filtering. When hybrid-partitioned, a filtered cache outperforms an unfiltered cache, showing that filtering can be beneficial by reducing the pressure at small partition sizes.

than Triangel.

Figure 14 shows that the removal of any of these components will affect Streamline's performance, so they are all crucial to Streamline's design. Since stream-based redundancy can often be benign (see Section V-C2), the removal of Streamline's metadata buffer or its stream alignment operation impacts performance less than the removal of tagged set-partitioning or TP-Mockingjay, which directly improve the utilization of the metadata store.

5) Partial Tag Aliasing: To enable 32-way set associativity per cache set, Streamline stores partial trigger tags in the tag array. If the partial tag of an entry X matches that of an existing entry Y, then X is placed in the same LLC way as Y so that metadata access only requires one LLC read. In other words, aliasing of partial tags can constrain and affect metadata replacement. In practice, we observe that by utilizing 6-bit partial tags, only 3.8% of the correlations in the metadata store alias. Thus, the effects on replacement are minimal. Moreover, each additional trigger bit further halves the amount of aliasing.

6) Filtering Prefetch Coverage Loss: While filtering can result in loss of prefetch coverage, Figure 15 shows that Streamline's realignment operation recovers 72% to 79% of the lost performance. However, because the amount of filtering grows with the maximal partition size, if Streamline were to support greater partition sizes, it would see worse prefetch coverage loss at its smaller partition sizes. We propose two solutions: (1) Instead of equally dividing the trigger addresses across LLC sets, *skewed indexing* biases the trigger-to-set mapping towards the sets allocated for smaller partition sizes, so skewed indexing reduces the amount of filtering at those partition sizes. (2) Instead of purely way-partitioning or set-partitioning the LLC, *hybrid-partitioning* combines the two approaches. For example, a 0.25MB metadata store would filter 75% of triggers under way-partitioning (2048 sets, 2 ways) and set-partitioning (512 sets, 8 ways), but hybrid-partitioning (1024 sets, 4 ways) would only filter 50% of triggers. Empirically, filtering reduces Streamline's performance by 1 to 1.8 percentage points for small partition sizes; however, we find that (1) skewed indexing enables Streamline to recover all of this performance loss, and (2) hybrid-partitioning enables Streamline to achieve higher performance than an unfiltered Streamline, demonstrating that a controlled amount of filtering can be beneficial for smaller partition sizes because it reduces pressure on the metadata store.

VI. Conclusion

In this paper, we have advanced the state-of-the-art in on-chip temporal prefetching by creating a metadata management scheme that leverages the semantics of a stream, and we have explained how our Streamline prefetcher addresses the various challenges that arise from a stream-based metadata representation.

Our new scheme has several benefits: (1) It simplifies the overall prefetcher design, (2) it eliminates redundancy in the metadata store, and (3) it more effectively manages the metadata store, which translates to significantly better prefetch coverage. When compared against Triangel, Streamline sees better prefetch coverage (+12.5 percentage points) and prefetch accuracy (+3.6 percentage points). In terms of performance, Streamline achieves 8.1% speedup over a system with a stride prefetcher, outperforming Triangel by 3 percentage points in a single-core setting and by 6.7 percentage points in a multi-core setting. Two points are noteworthy: First, Streamline matches the performance of Triangel even when Triangel is given twice the metadata storage; and second, while Triangel provides no benefit in a 8-core system with Berti in the L1D, Streamline provides an additional 3.8 percentage points of speedup.

Acknowledgments

We thank Akanksha Jain, Carson Molder, Molly O'Neil, and the anonymous referees for their insightful comments and suggestions on previous versions of this paper.

References

[1] "SPEC 2006," 2006. [Online]. Available: https://www.spec.org/cpu2006/
[2] "SPEC 2017," 2017. [Online]. Available: https://www.spec.org/cpu2017/
[3] "Sunny Cove microarchitecture: Going deeper and wider," 2019. [Online]. Available: https://www.anandtech.com/show/14514/examining-intels-ice-lake-microarchitecture-and-sunny-cove/3
[4] S. Ainsworth and L. Mukhanov, "Triangel: A high-performance, accurate, timely on-chip temporal prefetcher," in *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. IEEE, jul 2024, pp. 1202–1216. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00090
[5] G. Ayers, H. Litz, C. Kozyrakis, and P. Ranganathan, "Classifying memory access patterns for prefetching," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 513–526. [Online]. Available: https://doi.org/10.1145/3373376.3378498
[6] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Domino temporal data prefetcher," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 131–142.
[7] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 399–411.

[8] S. Beamer, K. Asanović, and D. Patterson, "The GAP benchmark suite," 2017. [Online]. Available: https://arxiv.org/abs/1508.03619

[9] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[10] R. Bera, K. Kanellopoulos, S. Balachandran, D. Novo, A. Olgun, M. Sadrosadati, and O. Mutlu, "Hermes: Accelerating long-latency load requests via perceptron-based off-chip load prediction," in *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '22. IEEE Press, 2023, p. 1–18. [Online]. Available: https://doi.org/10.1109/MICRO56248.2022.00015

[11] R. Bera, K. Kanellopoulos, A. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A customizable hardware prefetching framework using online reinforcement learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, oct 2021. [Online]. Available: https://doi.org/10.1145%2F3466752.3480114

[12] R. Bera, A. V. Nori, O. Mutlu, and S. Subramoney, "DSPatch: Dual spatial pattern prefetcher," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 531–544.

[13] R. Bera, A. Ranganathan, J. Rakshit, S. Mahto, A. V. Nori, J. Gaur, A. Olgun, K. Kanellopoulos, M. Sadrosadati, S. Subramoney, and O. Mutlu, "Constable: Improving performance and power efficiency by safely eliminating load instruction execution," in *Proceedings of the 51st Annual International Symposium on Computer Architecture*, ser. ISCA '24. IEEE Press, 2025, p. 88–102. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00017

[14] 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, p. 1–13. [Online]. Available: https://doi.org/10.1145/3307650.3322207

[15] J. Collins, S. Sair, B. Calder, and D. Tullsen, "Pointer cache assisted prefetching," in *35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002. (MICRO-35). Proceedings.*, 2002, pp. 62–73.

[16] Q. Duong, A. Jain, and C. Lin, "A new formulation of neural data prefetching," in *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. IEEE, 2024, pp. 1173–1187. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00088

[17] 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*, 2023, pp. 698–713.

[18] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jiménez, E. Teran, S. Pugsley, and J. Kim, "The Championship Simulator: Architectural simulation for education and competition," 2022.

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

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

[21] ——, "Back to the future: Leveraging Belady's algorithm for improved cache replacement," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 78–89.

[22] 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, p. 60–71. [Online]. Available: https://doi.org/10.1145/1815961.1815971

[23] S. Jamilan, T. A. Khan, G. Ayers, B. Kasikci, and H. Litz, "APT-GET: Profile-guided timely software prefetching," in *Proceedings of the Seventeenth European Conference on Computer Systems*, ser. EuroSys '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 747–764. [Online]. Available: https://doi.org/10.1145/3492321.3519583

[24] L. Jia, J. P. Mcmahon, S. Gudaparthi, S. Singh, and R. Balasubramonian, "PATHFINDER: Practical real-time learning for data prefetching," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 785–800.

[25] D. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 197–206.

[26] D. Joseph and D. Grunwald, "Prefetching using Markov predictors," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997, pp. 252–263.

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

[28] H. Litz, G. Ayers, and P. Ranganathan, "CRISP: critical slice prefetching," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 300–313. [Online]. Available: https://doi.org/10.1145/3503222.3507745

[29] A. Ltd., "Arm Cortex-A78AE core technical reference manual revision r0p1, CPUECTLR EL1, CPU extended control register, el1." [Online]. Available: https://developer.arm.com/documentation/101779/0001/Registerdescriptions/AArch64-system-registers/CPUECTLR-EL1--CPU-Extended-Control-Register--EL1

[30] ——, "Arm Cortex-X2 core technical reference manual r2p0, IMP CPUECTLR EL1, CPU extended control register." [Online]. Available: https://developer.arm.com/documentation/101803/0200/AArch64-system-registers/AArch64-generic-system-control-register-summary/IMPCPUECTLR-EL1--CPU-Extended-Control-Register

[31] ——, "Arm Neoverse N2 core technical reference manual r0p0, IMP CPUECTLR EL1, CPU extended control register." [Online]. Available: https://developer.arm.com/documentation/102099/0000/AArch64-registers/AArch64-genericsystem-control-registers/IMP-CPUECTLR-EL1--CPU-Extended-Control-Register

[32] P. Michaud, "Best-offset hardware prefetching," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 469–480.

[33] A. Naithani, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Vector runahead," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 195–208.

[34] A. Naithani, J. Roelandts, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Decoupled vector runahead," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 17–31.

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

[36] K. J. Nesbit and J. E. Smith, "Data cache prefetching using a Global History Buffer," *IEEE Micro*, vol. 25, no. 1, pp. 90–97, 2005.

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

[38] L. Peled, S. Mannor, U. Weiser, and Y. Etsion, "Semantic locality and context-based prefetching using reinforcement learning," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. Portland Oregon: ACM, Jun. 2015, pp. 285–297. [Online]. Available: https://dl.acm.org/doi/10.1145/2749469.2749473

[39] L. Peled, U. Weiser, and Y. Etsion, "A neural network prefetcher for arbitrary memory access patterns," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, oct 2019. [Online]. Available: https://doi.org/10.1145/3345000

[40] A. Pellegrini, "Arm Neoverse N2: Arm's 2nd generation high performance infrastructure CPUs and system IPs," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–27.

[41] S. H. Pugsley, Z. Chishti, C. Wilkerson, P.-f. Chuang, R. L. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, "Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 626–637.

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

[43] J. Roelandts, A. Naithani, S. Ainsworth, T. M. Jones, and L. Eeckhout, "Scalar vector runahead," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2024, pp. 1367–1381. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MICRO61859.2024.00101

[44] A. Roth and G. Sohi, "Effective jump-pointer prefetching for linked data structures," in *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, 1999, pp. 111–121.

[45] 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)*, 2022, pp. 558–572.

[46] M. Shakerinava, M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Multi-lookahead offset prefetching," in *3rd Data Prefetching Championship*, 2019.

[47] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 141–152.

[48] Z. Shi, A. Jain, K. Swersky, M. Hashemi, P. Ranganathan, and C. Lin, "A hierarchical neural model of data prefetching," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 861–873.

[49] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, p. 69–80, jun 2009. [Online]. Available: https://doi.org/10.1145/1555815.1555766

[50] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," *SIGARCH Comput. Archit. News*, vol. 34, no. 2, p. 252–263, May 2006. [Online]. Available: https://doi.org/10.1145/1150019.1136508

[51] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos, "Practical off-chip meta-data for temporal memory streaming," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, 2009, pp. 79–90.

[52] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi, "Temporal streaming of shared memory," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, ser. ISCA '05. USA: IEEE Computer Society, 2005, p. 222–233. [Online]. Available: https://doi.org/10.1109/ISCA.2005.50

[53] H. Wu, K. Nathella, M. Pabst, D. Sunwoo, A. Jain, and C. Lin, "Practical temporal prefetching with compressed on-chip metadata," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2858–2871, nov 2022.

[54] 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 Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 996–1008. [Online]. Available: https://doi.org/10.1145/3352460.3358300

[55] H. Wu, K. Nathella, D. Sunwoo, A. Jain, and C. Lin, "Efficient metadata management for irregular data prefetching," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 1–13.

[56] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "IMP: Indirect memory prefetcher," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 178–190.

[57] P. Zhang, A. Srivastava, A. V. Nori, R. Kannan, and V. K. Prasanna, "TransforMAP: Transformer for memory access prediction," in *The International Symposium on Computer Architecture (ISCA), ML for Computer Architecture and Systems Workshop*, 2021.

[58] Y. Zhang, N. Sobotka, S. Park, S. Jamilan, T. A. Khan, B. Kasikci, G. A. Pokam, H. Litz, and J. Devietti, "RPG2: Robust profile-guided runtime prefetch generation," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 999–1013. [Online]. Available: https://doi.org/10.1145/3620665.3640396