Performance-Driven Composite Prefetching with Bandits

Charles Block
coblock2@illinois.edu
University of Illinois
Urbana, Illinois, USA

Pedro Palacios Almendros
pedro5@illinois.edu
University of Illinois
Urbana, Illinois, USA

Abraham Farrell
af28@illinois.edu
University of Illinois
Urbana, Illinois, USA

Gerasimos Gerogiannis
gg24@illinois.edu
University of Illinois
Urbana, Illinois, USA

Josep Torrellas
torrella@illinois.edu
University of Illinois
Urbana, Illinois, USA

Abstract

Data prefetching is a complex problem. Its complexity has spurred research advancements for over half a century. Our submission to DPC4 is grounded on two observations. First, we observe that prefetchers are typically designed to optimize different low-level metrics such as timeliness, accuracy, and coverage. However, the correlation between those low-level metrics and the actual performance impact is complex and many times fluctuates depending on the application and system status. Second, we observe that industry often integrates ensembles of smaller prefetchers instead of large monoliths in the cache hierarchy of real products.

Motivated by those observations, we (1) split the available competition storage budget across different sub-prefetchers at different cache levels, and we (2) use *performance-driven* reinforcement-learning (RL) agents to manage and coordinate the activity of our composite prefetchers. Our RL agents are based on the Multi-Armed Bandit (MAB) model, and instead of optimizing for one or more low-level metrics, they are configured to directly maximize performance. Further, to improve the fairness of the prefetchers in multicore configurations, we apply a version of the Micro-MAMA arbitrator, which also relies on Multi-Armed Bandit forms of Reinforcement Learning. Compared to the original MAB and μMama works, we use a wider selection of lightweight prefetchers in our RL-managed ensemble at the L2, combined with Berti at the L1.

In this submission, we discuss the ensemble of prefetchers used in our design, as well as the coordination mechanisms used between them. Our Bandit-based composite prefetcher achieves strong single-core performance in the two system configurations evaluated in this competition, with geomean speedups of 2.7% and 1.2% over the baseline system in the Full- and Limited-Bandwidth cases, respectively. We conclude by discussing learnings we acquired during the design and tuning process that we believe can prove useful for the design space exploration and evaluation of future prefetchers.

1 Introduction

Over the past few decades, data-driven machine-learning (ML) algorithms have become widespread for a variety of modern applications. Importantly, such algorithms are also gaining momentum in computer architecture, with applications in branch prediction [19], prefetching [2, 5, 16], and more [4, 15]. Many works have used a particular subclass of ML called Reinforcement Learning (RL) to optimize the system's behavior in real time [2, 5, 7, 13]. A significant advantage of RL approaches is their ability to maximize a variety of high-level metrics such as the overall performance [2, 3, 5].

Many of the state-of-the-art prefetchers today such as Berti [11], are designed to optimize performance indirectly by targeting lower-level metrics such as accuracy, timeliness, and coverage. However, the relationship between such lower-level prefetch metrics and the main optimization target, which is the system's performance, is complex and non-linear. To make matters worse, this relationship often changes depending on the application and system state. For example, an increase in coverage from 85% to 90% can prove more detrimental for performance than an increase from 45% to 50%. Further, a slight increase in accuracy can be more profitable than a larger increase in coverage in a scenario where the main memory bandwidth is heavily utilized.

Alternatively, systems can use higher-level performance-driven action selection. RL agents that utilize the system's performance as the learning reward can make data-driven decisions that directly maximize performance. Further, recent core designs such as Intel Atom [8] employ an ensemble of simple prefetchers with runtime high-level configurability, instead of employing a large monolithic prefetcher. We believe that using simple RL agents to make performance-driven decisions about the behavior of these composite prefetchers is low-hanging fruit.

In this DPC4 submission, we present such an RL-driven composite prefetcher. Our RL control agent is based on the lightweight Micro-Armed Bandit (Bandit) [5], and instead of optimizing for one or more low-level metrics, it is configured to directly maximize performance. Further, to improve the fairness of our composite prefetchers in multicore configurations, we apply a version of the Micro-MAMA arbitrator [3], which similar to Bandit, relies on the Multi-Armed Bandit [9] Reinforcement Learning algorithms. In comparison to [5] and [3], we use more advanced (but still lightweight) sub-prefetchers for our DPC4 composite prefetcher.

In Section 2 we provide background on the Micro-Armed Bandit [5] and Micro-MAMA [3] works, and we analyze how we modified those designs for our submission. In Section 3, we evaluate our proposal. Finally, in Section 4 we discuss learnings we acquired while preparing our submission for DPC4.

2 System Components

In this section, we provide a brief overview of Micro-Armed Bandit [5] and μMama [3], the two designs central to our submission.

2.1 Micro-Armed Bandit

2.1.1 Multi-Armed Bandit Algorithms. In Multi-Armed Bandit (MAB) algorithms, an *arm* refers to a specific action available to the MAB

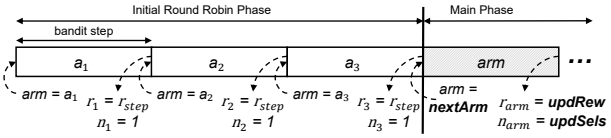


Figure 1: Overview of a MAB algorithm (from [6]).

RL agent, while a *bandit step* is defined as the time duration for which the agent is idle waiting to observe the outcome of its previous arm selection. r_{step} is the reward received at the end of a bandit step. For every arm i, two variables are needed: the average reward r_i that previous selections of this arm have yielded, and the number of times n_i that this arm has been selected in the past.

Figure 1 provides a general overview of a MAB algorithm. It begins with an initial round robin phase, during which all the arms are tried once. For each arm i, r_i is set to the r_{step} received during that arm's step, and n_i is set to 1. Then, the main phase of the algorithm begins, which lasts for as long as the agent keeps interacting with the environment. It consists of three basic functionalities, which depend on the specific MAB algorithm used. Those are: *nextArm()*, which selects the next arm to be tried; *updSels(arm)*, which updates the number of selections n_i for the currently selected arm i and potentially other arms; and *updRew*(r_{step}), which updates the reward r_i for the currently selected arm i after the bandit step is over and the r_{step} has been collected.

In the context of our work, different arms represent different configurations of our composite prefetcher (e.g. different sub-prefetchers turned on or off, and with different degrees). Further, we select the arm to try next using the *Discounted Upper Confidence Bound* (DUCB) algorithm (see [5]), which implements an intelligent arm exploration that accounts for both past rewards and selection frequencies of different arms. To adapt to dynamic scenarios, the DUCB algorithm progressively forgets past rewards.

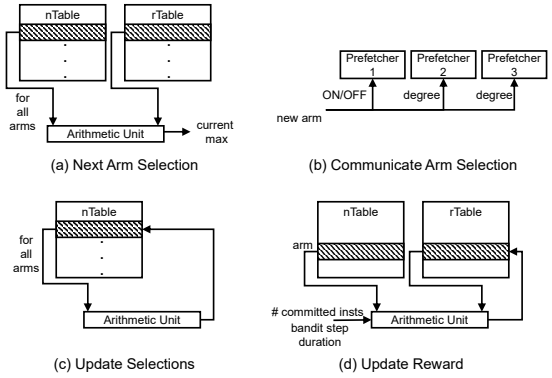


Figure 2: Micro-Armed Bandit microarchitecture (from [6]).

2.1.2 Bandit Agent. To coordinate our sub-prefetchers, we use the Bandit hardware agent as proposed in [5]. Bandit has two tables, an arithmetic unit, and some control logic. The two tables are the *nTable* and the *rTable*, and each has as many entries as the number of arms. For each arm i, the *nTable* contains the number of times that i has been selected so far (n_i), while the *rTable* contains the current value of its reward (r_i). The arithmetic unit executes the arithmetic operations in the *nextArm*, *updSels*, and *updRew* functions in hardware.

Figure 2(a) shows the implementation of *nextArm*. The hardware reads the nTable and rTable for all the arms, calculates the corresponding potentials, and picks one arm as the new arm. Then, in Figure 2(b), Bandit control logic communicates the arm selection to the controlled entity—in the figure, the L2 data prefetcher. In the background, Bandit updates the nTable (Figure 2(c)) according to the logic of function *updSels*. Once the bandit step is over, the Bandit arithmetic unit receives the appropriate hardware performance counters, computes the step's reward (r_{step}), and accumulates it into the rTable entry of the corresponding arm (Figure 2(d)). The figure assumes that the reward is the core's average IPC. This process repeats continuously.

2.2 Micro-MAMA

In multicore systems, each core and its associated prefetchers compete for resources such as memory bandwidth. As discussed in [3], this competition influences the learning process of RL prefetchers like Bandit that aim to maximize their own core's IPC, and leads to a reduction in their performance. Fundamentally, this issue stems from individual RL agents prioritizing their own performance over the rest of the system's. To alleviate this issue, for our submission we augment our local Bandit prefetching agents with a global µMama coordinator, as proposed in [3].

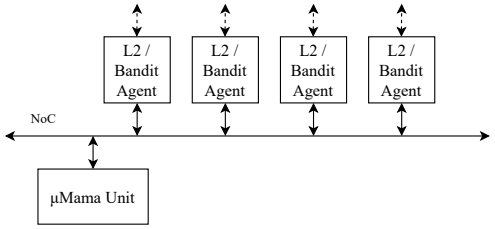


Figure 3: Local agents communicate with a µMama Unit.

2.2.1 µMama System. To solve the problem of conflicting interests, one cannot simply measure the total IPC and provide a single reward to all of the RL agents. Such an approach would result in poor credit assignment for good or bad actions taken by the individual agents. Instead, µMama takes a balanced approach, utilizing both greedy local agents and system-level action tracking. Inspired by prior theoretical work [12], locally-greedy Bandit agents in each private prefetcher explore promising actions, while a system-level agent monitors and overrides these agents to dictate system policy when it believes this will improve performance. In most cases, these local agents still use their local core's IPC as reward, whereas the system-level µMama supervisor is rewarded with an estimation of the whole system's performance.

Figure 3, shows distributed per-core Bandit agents and a global µMama Unit. The latter tracks high-performing "joint actions," formed by the combination of individual actions. Figure 4 shows the structure of the µMama Unit. The high-performing joint actions identified by the µMama Unit, together with their resulting IPC values are stored in a highly-associative structure called the *Joint Action Value (JAV) cache*. The µMama Unit also contains another RL agent, the µMama Arbiter, which determines, at each timestep, whether to allow the local Bandit agents to operate independently or to force on them the best-known joint action from its JAV cache.

The µMama Arbiter is itself a Bandit agent that uses the typical two tables (rTable and nTable as shown in Figure 4) to make its decision at every time step.

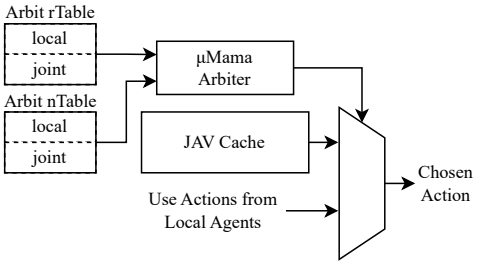


Figure 4: Structure of the µMama Unit. The Arbiter chooses between forcing a joint action taken from the JAV cache or letting the local agents use their actions on their own.

In this system, the local agents are primarily intended to support exploration of new joint actions, while the µMama Unit records (and sometimes initiates) high-performing combinations of individual actions. Rather than randomly sampling or iterating over the very large joint action space, local agents help focus the system's exploration on joint actions that have the potential to improve their core's performance.

2.2.2 The System-Level Reward. To use RL to optimize for system-level performance, one must be able to quantify that performance. µMama [3] was originally designed with the Weighted Speedup and Harmonic Speedup metrics in mind, where each core's speedup over a simple no-prefetcher system is arithmetically or harmonically averaged, respectively. However, for DPC4, a different variant of the harmonic mean speedup is being used, which measures each core's performance relative to a Berti [14]+Pythia [2] configuration. Although this may appear to be a similar metric, in practice it is much more difficult to estimate a given core's speedup over this new baseline than it is to estimate its speedup over a no-prefetcher system. To avoid the noise that these inaccurate estimations may cause, we choose to instead formulate µMama's reward as the geometric mean of the core IPCs. Although this is not entirely faithful to the metric that we wish to optimize, we observe that it correlates fairly well. Further, instead of only supplying the global reward to a small number of "low-importance" cores (see [3]), we find that it is beneficial to supply *every* agent with a reward derived partly from the whole system's performance and partly from its own core's performance; in our case, we use a 15%/85% split.

2.2.3 Communication and Latency. When coordinating several agents, it is beneficial to have a common timestep. To accomplish this, when a local agent reaches a threshold number of L2 demand accesses, it sends a message to µMama to mark itself as ready to advance. Once the majority of local agents do so, µMama broadcasts instructions to begin the next timestep. µMama hides most of this communication and the exchange of rewards between Bandits and the µMama Unit by planning one timestep ahead (see [3]).

2.3 Composite Prefetcher

We use Bandit and µMama to control a composite prefetcher at the L2. Our composite prefetcher includes the Next Line, Stride, and Stream prefetchers, similar to the original Bandit and µMama works.

Table 1: The components making up our system.

Level	Component	Controlled by Bandit/µMama?	Storage (KiB)	Total
L1D	Berti Pref.	No	29.45	29.45 KiB
L2	Next Line	Yes	0.00	
	Stride	Yes	12.02	
	Stream	Yes	5.63	
	SMS [17]	Yes	66.38	
	BOP [10]	Yes	6.23	
	Bloom Filter		0.24	
	Bandit	Agent	0.32	
	µMama	Agent	0.07	90.89 KiB
LLC	Nothing			0 KiB

However, we found that performance improved if we utilized the extra competition storage budget to also incorporate the SMS [17] and BOP [10] prefetchers in our ensemble. To prevent multiple prefetchers from issuing duplicate prefetches, we also use a Bloom filter to track the most recent accesses to the L2. At the L1, similar to the baseline, we use Berti, which operates independently and is configured as in [14]. We do not include any prefetcher at the LLC.

The storage requirements of our system are shown in Table 1. The local Bandits each maintain two tables to track the rewards of each arm, which are stored as double-precision floats, along with a few registers for tracking normalization factors and buffering communication with the central µMama agent. We utilize two capacity-50 bloom filters in a ping-pong fashion to reduce the number of redundant prefetches sent to the L2 cache; a 4-way, 128-set ip-stride prefetcher; a stream prefetcher with 256 trackers; and the SMS [17] and BOP [10] prefetchers. The central µMama unit only requires 37 bytes for the JAV cache, plus another 32 for its arbiter.

Table 2 shows the different arms (i.e. L2 sub-prefetcher configurations) that our Bandits can choose from. As discussed, our L2 Bandits are controlled by µMama. The Bandit/µMama hyperparameters used in our design (Table 3) are similar to [3].

Table 2: L2 Bandit arms used in our experiments. Each arm represents a configuration of the L2 prefetchers. All configurations use Berti at the L1 and no LLC prefetcher. A degree of 0 deactivates the Stride and Stream sub-prefetchers.

Arm ID	0	1	2	3	4	5	6	7	8
Next Line On?	✗	✗	✗	✓	✗	✗	✗	✗	✗
Stride Degree	0	0	0	0	3	0	3	5	5
Stream Degree	0	3	5	6	7	15	15	20	20
SMS On?	✗	✗	✗	✗	✗	✗	✗	✗	✓
BOP On?	✗	✗	✗	✗	✗	✗	✗	✗	✗

Arm ID	9	10	11	12	13	14	15	16	17
Next Line On?	✗	✗	✗	✓	✗	✗	✗	✗	✗
Stride Degree	0	0	0	0	3	0	3	5	5
Stream Degree	0	3	5	6	7	15	15	20	20
SMS On?	✗	✗	✗	✗	✗	✗	✗	✗	✓
BOP On?	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 3: The hyperparameters used for the learning process in this work. See [3, 5] for more details.

Component	Parameters
Local Agents	$c = 0.01$, $\gamma = 0.9995$, step = 800 L2 accesses
µMama Arbiter	$c = 0.1$, $\gamma = 0.995$, $T_{arbit} = 5$
µMama JAV	2 entries, $\gamma = 0.999$

3 Evaluation

3.1 Single-Core Evaluation

We evaluate our prefetcher ensemble with the 133 traces released by the competition ahead of the submission deadline, running each for 50M warmup instructions followed by 200M measured instructions. The traces come from four sources: SPEC, graph problems, AI/ML, and cloud workloads from Google. We use the FullBW and LimitBW system configurations specified by the competition. Figure 5 shows the results: By replacing Pythia in the L2 with this ensemble of prefetchers, we see significant speedup in three out of the four workload categories, totaling 2.67% and 1.16% in the FullBW and LimitBW configurations, or 3.01% and 1.33%, respectively, if the categories are given equal weighting.

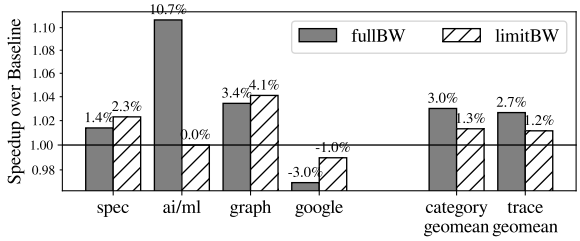


Figure 5: With a single core, our proposal outperforms the baseline system in three out of the four workload categories.

The AI/ML category appears to benefit greatly from aggressive streaming in the FullBW configuration (+10.69%), although it becomes entirely bandwidth bound in the LimitBW configuration. SPEC and the Graph workloads both see significant speedups as well, and actually observe more speedup relative to the Baseline in LimitBW than in FullBW, due to Bandit's ability to adapt its aggressiveness to the system. Although the Google workloads observe a slowdown relative to the baseline, we expect that this is not an issue with the Bandit learning process, but rather with the particular arms available to it. As we will discuss in Section 4, the art of selecting arms remains challenging.

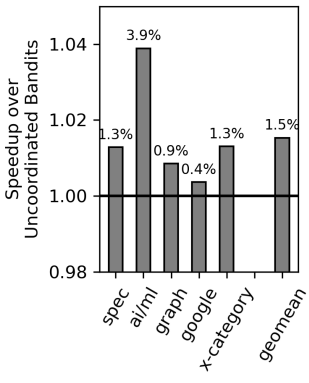
Most of the arms listed in Table 2 see regular use by our agents, but some are certainly more popular than others. In particular, we find that in the FullBW configuration, Bandit prefers Arms 17, 16, & 7 (being used about 15.1%, 9.5%, & 9.1% of the time, respectively). These all pair a very aggressive streamer with at least one other L2 sub-prefetcher. However, in the LimitBW configuration, we find that Bandit learns to use lower-aggressiveness arms, preferring Arms 0, 1, & 4 (25.4%, 10.3%, & 10.0%, respectively).

Notably, our design does not make use of an LLC prefetcher. This decision was made partly due to time constraints, since adding another prefetcher (or multiple) for a Bandit agent to control would have required a more in-depth design space exploration. However, although time and resources constrained our ability to evaluate an LLC prefetching ensemble, Table 4 shows the single-core results of placing *static* prefetchers at the LLC, along with our L2 ensemble. In all cases, we observe a performance degradation, ranging from -0.1% with a next-line prefetcher to -2.7% with an SMS prefetcher.

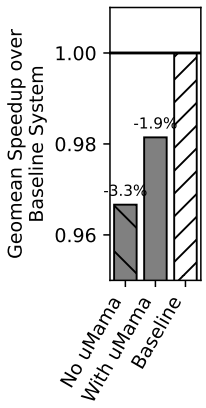
Table 4: Static prefetchers in the LLC degrade single core performance, compared to operating without one.

LLC Prefetcher	FullBW Speedup	LimitBW Speedup	Geomean
Next-Line	+0.0%	-0.3%	**-0.1%**
Berti	-1.6%	-2.2%	**-1.9%**
Pythia	-1.3%	-3.8%	**-2.6%**
SMS	-1.1%	-4.3%	**-2.7%**

3.2 Multi-Core Evaluation



(a) µMama performance vs uncoordinated Bandits.



(b) Bandit with and without µMama.

Figure 6: µMama provides better multicore performance than uncoordinated Bandits.

To evaluate the multicore performance, we follow the competition's method of randomly generating 50 workload mixes in each category, along with 50 mixes consisting of traces from all categories. We run each of these mixes until all cores have executed 250M instructions, and we report results for the first 250M.

As can be seen from Figure 6a, the µMama supervisor improves the performance of the system over that of uncoordinated Bandit agents in every category, with the greatest improvement being found in the AI/ML workloads. On average, this system outperforms an uncoordinated Bandit system by 1.5% when using the competition metric.

Despite this, as shown in Figure 6b, our ensemble underperforms the baseline system in the multicore case by 1.9% on average. Although our system still outperforms in the AI/ML workloads, the others fall short. We speculate that this is due to a sub-optimal selection of arms and hyperparameters for the multicore prefetching problem. We discuss this issue more in Section 4, but in short, time constraints and long simulations created difficulty in tuning our system for multicore performance before the competition deadline. A selection of more low-to-moderate aggressiveness arms, alternate µMama hyperparameters, and a reward function for µMama that more closely matched the evaluation criteria may have provided

better performance. Nonetheless, the improvement provided by the μMama system over uncoordinated Bandits, even without a full parameter exploration, demonstrates the importance of such a coordination system when using performance-driven agents.

4 Conclusion and Learnings

At the heart of our DPC4 submission lies a composite L2 prefetcher that offers high-level configurability. The composite prefetcher is controlled by a reinforcement learning based system that includes a private Bandit agent at each core's L2 and a global μMama coordinator. The Bandit and μMama agents make high-level configuration decisions with the goal of directly optimizing performance. We now discuss some learnings we acquired while preparing our DPC4 submission, that we believe can prove useful for future work.

1. The initial static selection of Bandit arms is crucial for the final performance of the system. However, the problem of design space exploration remains difficult. A Bandit-based system can only be as good as the sum of its individual arms – formed in our case by the sub-prefetchers of our composite L2 prefetcher. There is a very rich space in different sub-prefetcher configurations that one may choose from. Such configurations may not only differ in the activated prefetcher types and degrees but also on aspects such as cross-page prefetching [18], L1 prefetching on virtual or physical addresses, and more. At the same time, allowing for a large number of arms that the Bandit can choose from at runtime may decrease performance due to increased exploration cost. While preparing our submission, we attempted to use autotuning tools [1] with techniques such as evolutionary algorithms and gaussian optimization to programmatically derive a good set of arms. However, both of those methods did not produce good results, due to the huge design space and long simulation times. Instead, we manually picked the set of arms in Table 2, with humans always in the loop. Overall, we believe that Bandit-based solutions still have a large performance headroom, and we hope that *combining microarchitectural insights with autotuning methods such as gaussian optimization* to determine the action space may prove fruitful in the future.

2. The performance of Bandit and μMama is sensitive to their hyperparameters, and the optimal hyperparameters are sensitive to the underlying system configuration. Organizing fair head-to-head evaluations in a competition format is difficult, since simulation infrastructure must be bug-free and evaluation methodology must be carefully constructed. In the case of DPC4, this resulted in various updates to infrastructure and rules throughout the submission period to improve the quality of the competition. We found that after each of these changes, we would often find that a previously high-performing set of arms no longer worked as well as before, or that hyperparameters needed to be re-tuned. Although we found that autotuning tools tended to work better for numerical hyperparameters than arm selection, the long simulation times still restricted our design space exploration. In fact, due to the long-running nature of multicore simulations and some late competition changes, we were only able to test a handful of possible configurations of μMama before the submission deadline, and only with much-reduced instruction counts. This was one reason for the decision to use a simple geomean-based reward instead of attempting to accurately estimate the true speedup-over-baseline at runtime. We believe that the high sensitivity of performance to hyperparameters is a weakness of Bandit-based solutions that can motivate more research in the future.

3. Composite prefetchers as baselines in prefetching papers. As discussed in Section 1, industry is moving towards composite prefetchers. We believe that such systems, augmented with smart, potentially performance-driven coordinators, should become baselines for comparison even for standalone (not composite) prefetcher designs. Although we tried different prefetchers in our L2 ensemble of Table 1, we found that only SMS and BOP were able to raise performance beyond the ensembles used in the original Bandit/μMama works. To that end, we believe it is interesting to test how the performance of new prefetching proposals compares against iso-storage composite prefetchers, or whether such new proposals bring value when used as components in composite prefetchers.

Acknowledgments

This work was supported by NSF with grants CCF 2107470, CCF 2316233, and Graduate Research Fellowship DGE 21-46756; by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA; and by the IBM-Illinois Discovery Accelerator Institute. This work made use of computing resources of Illinois Computes, which is supported by the University of Illinois Urbana-Champaign (UIUC); and the Illinois Campus Cluster, operated in conjunction with NCSA.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[2] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 1121–1137. doi:10.1145/3466752.3480114

[3] Charles Block, Gerasimos Gerogiannis, and Josep Torrellas. 2025. Micro-MAMA: Multi-Agent Reinforcement Learning for Multicore Prefetching. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*. Association for Computing Machinery, New York, NY, USA, 884–898. doi:10.1145/3725843.3756096

[4] Abdoulaye Gamatié, Xin An, Ying Zhang, An Kang, and Gilles Sassatelli. 2019. Empirical model-based performance prediction for application mapping on multicore architectures. *Journal of Systems Architecture* 98 (2019), 1–16. doi:10.1016/j.sysarc.2019.06.001

[5] Gerasimos Gerogiannis and Josep Torrellas. 2023. Micro-Armed Bandit: Lightweight & Reusable Reinforcement Learning for Microarchitecture Decision-Making. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 698–713. doi:10.1145/3613424.3623780

[6] Gerasimos Gerogiannis and Josep Torrellas. 2024. Practical online reinforcement learning for microprocessors with micro-armed bandit. *IEEE Micro* 44, 4 (2024), 80–87.

[7] Yan Huang and Zhanyang Wang. 2024. RLOP: A Framework Design for Offset Prefetching Combined with Reinforcement Learning. In *Proceedings of the 13th International Conference on Computer Engineering and Networks*, Yonghong Zhang, Lianyong Qi, Qi Liu, Guangqiang Yin, and Xiaodong Liu (Eds.). Springer Nature Singapore, Singapore, 90–99.

[8] Intel Corporation. 2023. *Hardware Prefetch Controls for Intel® Atom® Cores*. Whitepaper 357930-001US. Intel Corporation. https://www.intel.com/content/www/us/en/content-details/795247/hardwareprefetch-controls-for-intel-atom-cores.html

[9] Tor Lattimore and Csaba Szepesvári. 2020. *Bandit Algorithms*. Cambridge University Press.

[10] Pierre Michaud. 2016. Best-offset hardware prefetching. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 469–480.

[11] Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals-Yúfera, and Alberto Ros. 2022. Berti: an accurate local-delta data prefetcher. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 975–991.

[12] Santiago Ontañón. 2017. Combinatorial multi-armed bandits for real-time strategy games. *J. Artif. Int. Res.* 58, 1 (Jan. 2017), 665–702.

[13] Leeor Peled, Shie Mannor, Uri Weiser, and Yoav Etsion. 2015. Semantic locality and context-based prefetching using reinforcement learning. *SIGARCH Comput. Archit. News* 43, 3S (jun 2015), 285–297. doi:10.1145/2872887.2749473

[14] Alberto Ros. 2019. Berti: A per-page best-request-time delta prefetcher. *The 3rd Data Prefetching Championship* (2019).

[15] Subhash Sethumurugan, Jieming Yin, and John Sartori. 2021. Designing a Cost-Effective Cache Replacement Policy using Machine Learning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 291–303. doi:10.1109/HPCA51647.2021.00033

[16] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. 2021. A hierarchical neural model of data prefetching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 861–873. doi:10.1145/3445814.3446752

[17] Stephen Somogyi, Thomas F Wenisch, Anastassia Ailamaki, Babak Falsafi, and Andreas Moshovos. 2006. Spatial memory streaming. *ACM SIGARCH Computer Architecture News* 34, 2 (2006), 252–263.

[18] Georgios Vavouliotis, Marti Torrents, Boris Grot, Kleovoulos Kalaitzidis, Leeor Peled, and Marc Casas. 2025. To cross, or not to cross pages for prefetching?. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 188–203.

[19] Siavash Zangeneh, Stephen Pruett, Sangkug Lym, and Yale N. Patt. 2020. BranchNet: A Convolutional Neural Network to Predict Hard-To-Predict Branches. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 118–130. doi:10.1109/MICRO50266.2020.00022