An Online Machine Learning Based Prefetch Selection Controller in gem5

M Sadman Sakib
Department of Electrical and Computer Engineering
University of Wisconsin–Madison
Madison, Wisconsin, USA
msakib@wisc.edu

Munasib Ilham
Department of Electrical and Computer Engineering
University of Wisconsin–Madison
Madison, Wisconsin, USA
ilham@wisc.edu

***Abstract*—Hardware data prefetching is a widely used technique for reducing effective memory latency, but no single prefetcher performs optimally across all workloads and cache configurations. Aggressive prefetchers may improve performance for regular memory access patterns, while causing cache pollution and bandwidth contention for irregular workloads. This work presents an online machine learning based prefetch selection controller implemented in the gem5 simulator that dynamically chooses among multiple heterogeneous hardware prefetchers at runtime. Our design formulates prefetcher selection as a contextual bandit problem and employs a lightweight tabular reinforcement learning algorithm to select the most suitable prefetcher at fixed execution epochs. The controller monitors runtime performance indicators such as IPC and L2 miss behavior to guide adaptation. We evaluate the controller across the MachSuite benchmark suite under multiple cache configurations and compare its performance against static prefetching policies, including stride-based and tagged prefetchers, as well as prefetching disabled. Experimental results show that while a single static prefetcher often performs best for a specific workload, the proposed ML-based controller consistently avoids worst case behavior and typically matches or approaches the second-best static configuration across workloads and cache sizes. We analyze why prefetch accuracy alone does not correlate directly with performance, explain why the ML controller rarely outperforms the best static prefetcher, and discuss how the design could be extended to improve performance further. Overall, this work demonstrates the practicality and robustness of online learning for adaptive prefetch control in realistic microarchitectural settings.**

I. Introduction

Modern processors increasingly rely on hardware data prefetching to hide long memory latencies. As memory hierarchies grow deeper and workloads become more diverse, effective prefetching plays a crucial role in achieving high performance. However, prefetching is fundamentally a speculative optimization: issuing incorrect or mistimed prefetches can degrade performance by polluting caches, consuming memory bandwidth, and increasing contention for miss handling resources. A large body of prior work demonstrates that no single prefetcher is universally optimal. Simple stride prefetchers perform well on regular streaming workloads, while correlation-based prefetchers such as tagged or Markov prefetchers are more effective for complex access patterns. In practice, processors must choose between prefetching policies without prior knowledge of application behavior. This project investigates whether online machine learning can be used to dynamically select among multiple prefetchers at runtime. Rather than designing a new predictive prefetcher, we focus on prefetcher selection, treating each prefetcher as a candidate for action and learning which one to apply based on observed performance feedback. We implement a contextual bandit-based controller in gem5 that selects between multiple existing prefetchers at fixed execution epochs. The controller is lightweight, online, and does not rely on offline training or application profiling. Our evaluation shows that while the controller does not always outperform the best static prefetcher, it provides robust performance across workloads and cache configurations and avoids pathological slowdowns.

II. Background

A. Hardware Prefetching

Hardware prefetchers attempt to predict future memory accesses and fetch data into the cache before it is requested by demand accesses. Common classes of prefetchers include:

- Stride Prefetchers, which detect fixed-distance access patterns and issue prefetches at a constant stride.
- Correlation-Based Prefetchers (Tagged Prefetchers), which record access histories and predict future addresses based on observed correlations.
- Pointer-Chasing Prefetchers, which follow dependent load chains. While prefetchers can significantly reduce cache miss latency, they introduce several costs:
- Cache pollution from unused prefetched blocks
- Memory bandwidth consumption
- MSHR contention and prefetch timeliness issues

These trade-offs become especially severe in systems with small caches or limited memory-level parallelism.

B. Reinforcement Learning and Contextual Bandits

Reinforcement learning (RL) models decision-making problems where an agent interacts with an environment, takes actions, and receives rewards. However, full RL formulations such as Markov Decision Processes are often too complex for microarchitectural control. A contextual bandit simplifies RL by:

- Ignoring long-term state transitions

- Selecting an action based only on the current context
- Receiving an immediate reward

This makes contextual bandits well-suited for runtime hardware control, where decisions must be fast, lightweight, and stable.

C. Why Bandits for Prefetch Selection

Prefetch selection is naturally a bandit problem:

- Each prefetcher is an independent "arm"
- The reward (performance improvement) is observable within a short time window
- Long-horizon planning is unnecessary

III. Design Overview

A. High-Level Architecture

The ML-based prefetch controller sits at the L2 cache level in gem5 and manages a set of child prefetchers. At any given epoch, exactly one child prefetcher is active, ensuring isolation of effects and clean attribution of performance outcomes.

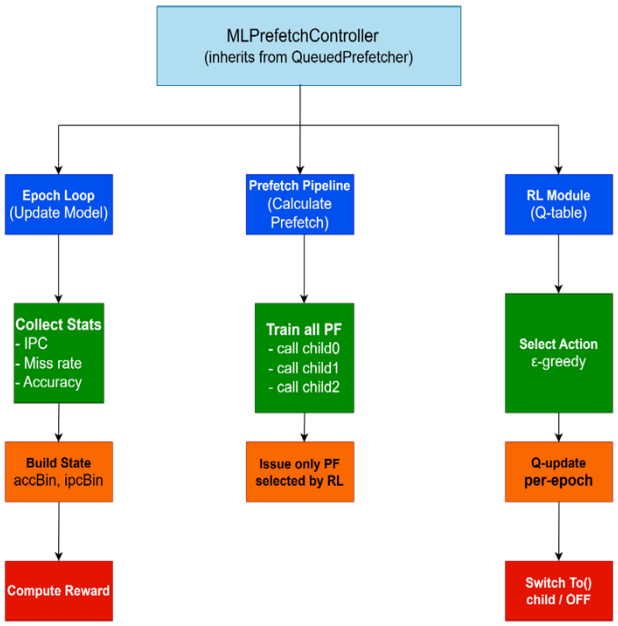

Fig. 1. ML Prefetch Controller's core components.

B. Epoch-Based Control

Execution is divided into fixed-length epochs. At the end of each epoch, the controller: 1. Collects runtime statistics 2. Computes a reward 3. Updates its Q-table 4. Selects the next prefetcher This design ensures low overhead and stable adaptation.

IV. RL Formulation

A. State Representation

The state includes coarse-grained runtime features: Change in IPC, Change in L2 miss rate, Prefetch accuracy bucket. Each feature is discretized into a small number of bins, producing a compact state space.

B. Action Space

Actions correspond to enabling one of the prefetchers from the children's set. Any prefetcher can be added to the children's set through the configuration file. For our benchmarks we use the following children:

- Stride (1,1) [Action 0]
- Stride (4,2) [Action 1]
- Tagged prefetcher [Action 2]
- Prefetch OFF [Action 3]

Only one action is active per epoch.

C. Reward Function

The reward is a weighted combination of IPC improvement and Prefetch accuracy regularization. Accuracy is intentionally used as a soft penalty, not a primary objective. This explains why ML accuracy can be lower than static prefetchers while IPC remains competitive.

D. Learning Algorithm

We use tabular Q-learning with an ϵ-greedy policy:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r - Q(s,a)) \tag{1}$$

where $Q(s,a)$ is the estimated long-term return (action-value) starting from state s and taking action a, α is the learning rate controlling how much the Q-value is updated, and r is the immediate reward received after taking action a in state s.

Exploration decays over time, and learning can be disabled for final evaluation runs.

E. Relationship to Hiebel, Brown, and Wang

Our work is inspired by the contextual-bandit formulation introduced by Hiebel, Brown, and Wang [2], but differs in key ways:

TABLE I
Difference of Implementation

Aspect	*Prior Work*	*This Work*
Control target	Prefetch aggressiveness	Prefetcher selection
Scope	Single prefetcher	Multiple heterogeneous prefetchers
Reward	Coverage-oriented	IPC-driven
Implementation	Hardware	gem5 simulation

Thus, while the algorithmic foundation is similar, the control problem and design goals are fundamentally different.

V. gem5 Implementation

The controller is implemented as a custom `MLPrefetchController` SimObject that inherits from `QueuedPrefetcher` in gem5. It interfaces with the L2 cache for miss statistics, the CPU for IPC measurement and child prefetchers for candidate generation.

Prefetch attribution is tracked explicitly to compute per-prefetcher usefulness and redundancy statistics.

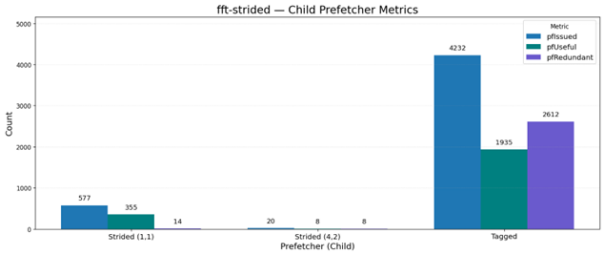

Fig. 2. Per-child prefetch statistics collected by the ML controller.

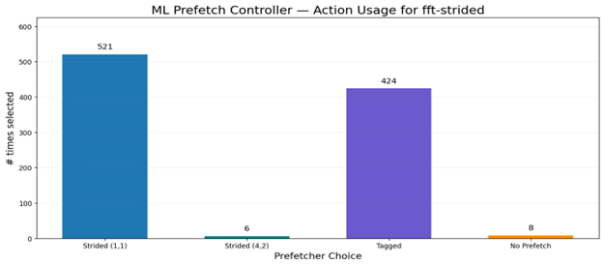

Fig. 3. Number of epochs each child prefetcher was selected by the controller.

The controller can be used fully online without any prior runs on the program. It can also be trained by running similar workloads multiple times to warm up the Q-table and then frozen by disabling exploration and learning for final evaluation. The Q-table can optionally be saved and reused across runs.

VI. Experimental Methodology

A. Benchmarks

We evaluate using the MachSuite benchmark suite, which includes a diverse set of memory behaviors, including dense linear algebra, graph traversal, and stencil computations. For the Machsuite benchmarks, the ml prefetch controller was run 3 times on the whole benchmark suite to warm up its average Q-table and then froze to run on each program/kernel for final benchmark readings.

B. Cache and CPU Configurations

We evaluate the proposed controller across multiple cache configurations. For L1 caches, we consider sizes of 32 KB with 8-way associativity (for MachSuite benchmarks) and 16 KB with 4-way associativity. For L2 caches, configurations include 256 KB with 8-way associativity (MachSuite) and 32 KB with 4-way associativity. All experiments use the same CPU and memory system parameters: an out-of-order, pipelined X86 O3CPU running at a 4 GHz clock frequency, paired with HBM-2 (2000 MT/s, 4-High, 1×64-bit) memory. This setup allows us to study the controller's performance under varying cache sizes while keeping the underlying processor and memory behavior consistent.

C. Baselines

The performance of the proposed ML-based controller is compared against several baseline configurations. These include no prefetching, which represents the system's baseline performance without any speculative memory accesses, as well as standard hardware prefetchers such as Stride (1,1), Stride (4,2), and the Tagged prefetcher. These baselines provide a spectrum of prefetching strategies, ranging from minimal speculative behavior to more aggressive schemes, allowing us to evaluate both the effectiveness and adaptability of the ML controller.

D. Metrics

We evaluate the proposed controller using several performance and behavioral metrics. Instruction per cycle (IPC) is measured to assess overall system performance. Cache miss rates indicate how effectively the controller reduces memory stalls. Prefetch behavior is characterized by accuracy (fraction of useful prefetches) and coverage (fraction of cache misses that were prefetched). We further examine prefetch usefulness and redundancy, which capture whether prefetches actually contribute to performance or lead to unnecessary memory traffic. Finally, action selection frequencies are analyzed to understand the controller's decision-making patterns and adaptation across different program phases.

E. Training Protocol and Evaluation Procedure

The ML-based prefetch controller is designed to operate in a fully online manner and does not require any prior offline profiling. However, to study the effect of training stabilization, we evaluate two modes of operation:

- **Online Mode**: The controller starts with an empty Q-table and learns continuously during program execution using fixed learning and exploration rates.
- **Train-and-Freeze Mode**: The controller is first run on a set of workloads multiple times to warm up the Q-table. Learning rate and exploration rate are then set to zero for a final evaluation run.

For MachSuite benchmarks, the controller was trained for three full passes across the benchmark suite to stabilize average Q-values. The final evaluation run was performed with learning and exploration disabled to ensure deterministic behavior and fair comparison against static prefetchers.

It is important to note that even in the train-and-freeze mode, the controller does not encode explicit application identifiers or phases. As a result, training primarily reduces noise in reward estimation rather than producing a fully specialized per-application policy. This design choice intentionally favors generality and robustness over application-specific optimization.

VII. Results and Analysis

A. Overall IPC Trends

Across MachSuite, the tagged prefetcher often achieves the highest IPC. It was also due to our cache size was relatively large for the Machsuite benchmarks, which does not penalize tagged prefetcher's aggressive prefetching enough for cache pollution. The ML controller typically ranks second-best and rarely performs worst.

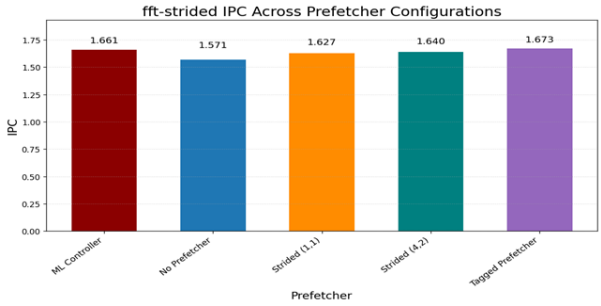

Fig. 4. IPC comparison for FFT-strided.

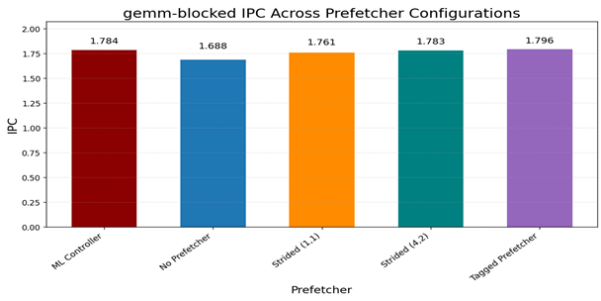

Fig. 5. IPC comparison for GEMM blocked.

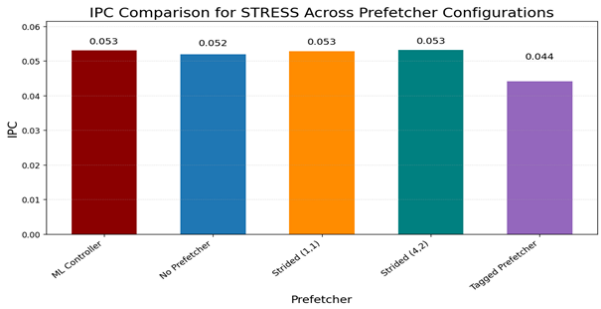

Fig. 6. IPC comparison for a custom test intended to penalize cache pollution.

We also benchmarked a custom stress workload consisting of a streaming friendly phase and a prefetcher unfriendly phase. As expected, tagged prefetcher performed worst due to polluting the cache with unnecessary prefetches. Both stride prefetchers did well as their benefit in the streaming friendly phase outweighed their penalty in prefetcher unfriendly phase. ML controller matches the performance of the stride prefetchers.

B. Accuracy vs. Coverage

Accuracy is defined as the ratio of useful prefetches to the total number of issued prefetches:

$$\text{Accuracy} = \frac{\sum \text{useful prefetches}}{\sum \text{issued prefetches}} \times 100\ (\%) \tag{2}$$

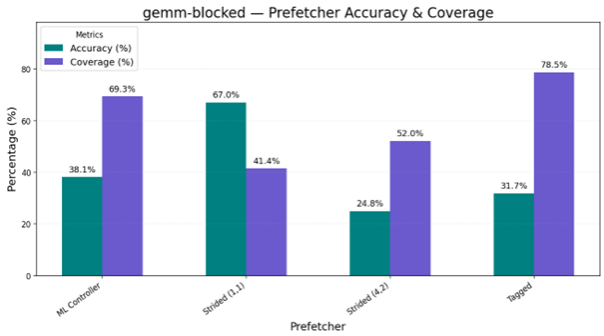

Fig. 7. ML Controller trades slightly higher accuracy with slightly lower coverage compared to best prefetcher (Tagged Prefetcher).

Coverage is defined as the fraction of L2 cache misses that are satisfied by prefetches issued by the prefetcher:

$$\text{Coverage} = \frac{\sum \text{L2 misses covered by prefetcher}}{\sum \text{L2 misses}} \times 100\ (\%) \tag{3}$$

High accuracy does not necessarily imply high performance. Some aggressive prefetchers achieve low accuracy but high coverage, which can still improve IPC due to the increased number of timely prefetches. This explains why the ML controller may prefer prefetchers with lower accuracy but better overall performance impact. Overall, the ML controller achieves a balanced middle ground: slightly higher accuracy than the most aggressive static prefetcher and slightly lower coverage than the most aggressive (or highest-coverage) prefetcher.

C. Action Selection Behavior and Policy Stability

To better understand the behavior of the learned policy, we analyze the frequency with which each prefetcher is selected during execution. Action selection statistics reveal that the ML controller does not converge to a single dominant prefetcher in most workloads. Instead, it distributes selections across multiple actions, occasionally disabling prefetching entirely.

This behavior is expected for three reasons. First, the controller operates under an ϵ-greedy policy during online learning, which explicitly enforces exploration. Second, the reward signal is noisy due to microarchitectural overlap, memory-level parallelism, and delayed effects of prefetching. Third, workloads often exhibit mixed or phase-varying access patterns where no single prefetcher is uniformly optimal.

As a result, even after training, the controller may continue to occasionally select suboptimal actions. While this can slightly reduce peak IPC compared to the best static prefetcher, it prevents catastrophic slowdowns and allows the controller to remain responsive to changing execution behavior. This conservative strategy reflects a deliberate design trade-off between peak performance and robustness.

D. Why ML Rarely Beats the Best Static Prefetcher

The proposed ML-based controller operates in an online, workload-agnostic, and exploration-aware manner, which fundamentally limits its ability to consistently outperform the

best static prefetcher on a per-workload basis. Unlike static prefetchers that are carefully tuned with full hindsight of workload behavior, the ML controller must learn and adapt during execution while balancing exploration and exploitation. This online learning requirement introduces conservative decision-making and occasional suboptimal actions, particularly during phase transitions. Consequently, while a static prefetcher may achieve higher peak performance for a specific workload, the ML controller intentionally trades peak performance for robustness and generality, aiming to deliver stable improvements across diverse workloads without prior knowledge or offline tuning.

E. Training Effects

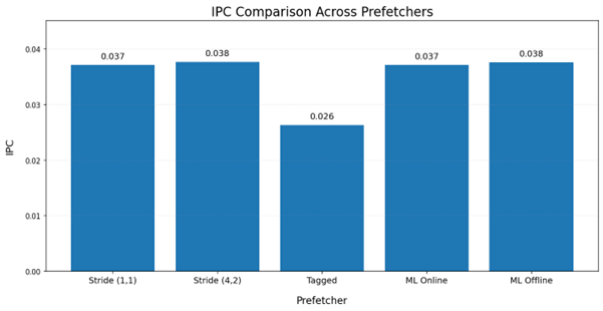

Fig. 8. Online ML controller almost matches the performance of best prefetcher, Stride (4,2). Using training and freeze (ML Offline) improves the ML controller slightly to match the performance of best prefetcher. Benchmarked on a custom program with smaller cache size.

The online ML controller (blue bars) almost matches the performance of the best static prefetcher, Stride (4,2). When trained and then frozen (ML Offline), the controller sees only a slight performance improvement, reaching parity with the best prefetcher. These results indicate that training on the same workload yields minimal gains, as the controller is primarily designed for online adaptation; training mainly smooths out transient noise rather than enabling effective workload profiling. Moreover, training on one type of workload and evaluating on a significantly different workload can sometimes degrade performance, reflecting the controller's sensitivity to workload shifts. This behavior is expected due to the absence of a phase oracle, coarse-grained state representation, and the effects of exploration penalties, which collectively limit the ability of offline training to substantially improve performance.

F. Why Prefetch Accuracy Is Low Yet Performance Remains Competitive

One recurring observation across experiments is that the ML-based controller exhibits lower prefetch accuracy than some static prefetchers, even on workloads included in the training set. This behavior may appear counterintuitive at first glance but is consistent with how accuracy is measured and how the controller is optimized.

Prefetch accuracy in gem5 is defined as the fraction of issued prefetches that are later used by demand accesses. This metric penalizes aggressive prefetchers that issue speculative requests, even when those requests successfully overlap memory latency or increase memory-level parallelism. Consequently, accuracy alone does not fully capture the performance impact of a prefetcher.

The ML controller optimizes for IPC-based reward rather than accuracy. As a result, it may select actions that generate additional speculative prefetches if they help hide latency or reduce stall cycles, even if many of those prefetches are ultimately unused. This effect is amplified in small-cache configurations, where timely prefetches can improve performance despite higher eviction and redundancy rates.

Furthermore, because the controller switches between different prefetchers across epochs, attribution of usefulness is fragmented across actions, further reducing measured accuracy. Therefore, low accuracy does not indicate poor learning; instead, it reflects a deliberate optimization choice aligned with end-to-end performance rather than local prediction quality.

VIII. Discussion and Limitations

A. Low ML Accuracy

The observed low machine learning prediction accuracy is an expected outcome of the design choices made in this work. First, the system dynamically switches between multiple prefetchers, resulting in non-stationary behavior that makes accurate prediction inherently more difficult. Second, the reward function prioritizes instruction per cycle (IPC) improvement rather than traditional accuracy metrics, allowing the model to favor aggressive or speculative prefetching strategies that may not always produce correct predictions but still improve overall performance. Finally, the model does not impose an explicit penalty for cache pollution or memory bandwidth usage, which can further reduce measured accuracy while still providing net performance gains. As a result, prediction accuracy alone is not a sufficient indicator of effectiveness for the proposed approach, and performance-oriented metrics are more appropriate for evaluating its impact.

B. Online Nature of the Algorithm

The proposed controller is inherently designed for online adaptation rather than convergence to a single static optimal policy. Program memory behavior is highly dynamic, with workload phases and access patterns changing throughout execution. As a result, continuously updating the policy allows the controller to respond to these changes in real time. This design choice explains why freezing the learning process does not yield a dramatic performance improvement over online adaptation, as a fixed policy may quickly become suboptimal when execution characteristics shift. Therefore, sustained online learning is a fundamental aspect of the controller's effectiveness rather than a limitation.

C. Online vs Offline Learning Trade-offs

Although the controller supports training and freezing, it is fundamentally designed as an online learning system rather than an offline optimizer. Offline training assumes stationary

behavior and stable phase boundaries, which are rarely present in realistic applications. In contrast, online learning allows the controller to continuously adapt to runtime behavior, resource contention, and phase transitions.

Our experiments show that freezing the controller after training yields only modest performance improvements. This outcome is expected, as training primarily stabilizes reward estimates rather than enabling perfect specialization. In some cases, freezing the policy can even reduce robustness when execution characteristics differ from the training runs.

Therefore, the ML controller should not be interpreted as a replacement for highly tuned static prefetchers. Instead, it serves as a general-purpose adaptive mechanism that performs well without requiring application-specific knowledge or offline tuning. This distinction is critical when evaluating its effectiveness and potential deployment scenarios.

D. Why the Design Is Still Useful

Despite the limitations discussed above, the proposed ML-based controller remains practically useful. By dynamically selecting among prefetching strategies, it helps avoid worst-case performance scenarios that can arise from poorly matched static prefetchers. The controller also adapts to cache and memory system constraints, enabling more resilient behavior under varying contention and resource pressure. Moreover, the design provides a flexible framework for extensibility, allowing future enhancements such as richer state representations, alternative reward formulations, or tighter hardware integration without requiring a complete redesign. Together, these properties make the approach a robust and extensible foundation for adaptive prefetching.

IX. Related Work

In this section, we describe the prior studies that have been made in the context of of development of ML-based Cache Prefetchers. Adaptive prefetching has been explored in commercial systems. Jiménez et al. demonstrate a dynamic prefetch reconfiguration mechanism on IBM POWER7 that adjusts prefetch depth and stride parameters at runtime and achieves measurable performance gains compared to static settings [1]. Their results confirm that fixed prefetch configurations are frequently suboptimal. In contrast, our work introduces a software-based adaptive controller within the gem5 simulation framework rather than modifying hardware mechanisms or microcode.

Hiebel and Brown model prefetch control as a contextual bandit problem, enabling per-epoch tuning of aggressiveness based on observed behavior [2]. Their formulation directly informs the algorithmic structure of our controller. Unlike their hardware-driven implementation, which operates across multiple prefetchers, our approach provides a reusable simulation infrastructure to support ML-based decision-making in gem5.

Bandit-based decision frameworks have also been applied to broader architectural resource management. Glassner and Crammer show that multi-armed bandits can dynamically optimize cache allocation policies under uncertainty [3]. Although their work focuses on thread-level cache resource partitioning rather than prefetching, it reinforces the suitability of online learning as a lightweight, hardware-friendly control mechanism. This supports our decision to adopt a contextual-bandit model for runtime prefetch modulation.

Lu et al. propose APAC, an adaptive prefetch framework that integrates the Pure Prefetch Coverage (PPC) metric to account for memory concurrency effects [4]. APAC demonstrates improved performance over prior adaptive mechanisms by distinguishing pure misses from overlapped memory accesses, revealing that traditional accuracy- and coverage-based metrics may misrepresent prefetch utility. While our current controller does not incorporate concurrency-sensitive features such as MSHR pressure or pure-miss tracking, APAC motivates future extensions in this direction.

Yang et al. introduce PAIC, a machine-learning-based cache replacement policy that distinguishes between demand and prefetch requests to reduce pollution and improve reuse prediction [5]. Although PAIC targets replacement rather than prefetch selection, it highlights the importance of cross-layer interactions between prefetching and cache management. Our controller may similarly expose dynamic prefetch state to replacement policies in later phases.

Prior ML-based prefetch selection techniques demonstrate the benefits of lightweight learning. Rahman et al. and Alcorta et al. show that selecting prefetch configurations based on application characteristics can outperform static policies. Our work differs by performing fine-grained online learning at runtime rather than per-application or offline selection.

Neural predictive prefetchers such as Voyager and Twilight improve prediction accuracy by capturing complex temporal and spatial correlations. However, these approaches require substantial storage and computational overhead, making them less practical for hardware-realistic integration. Accordingly, our controller intentionally avoids neural inference in gem5 and instead focuses on lightweight, feedback-driven control mechanisms that can feasibly translate to future hardware implementations.

X. Threats to Validity

Several factors may influence the generality of our results. First, all experiments were conducted in the gem5 simulator, which, while widely used, may not perfectly capture all timing and contention effects present in real hardware. Second, our state representation is intentionally coarse-grained to maintain low overhead. Richer state features could enable more precise control but may introduce complexity and instability.

Third, the reward function prioritizes IPC improvement without explicitly modeling memory bandwidth usage, energy consumption, or fairness. As a result, the controller may favor aggressive behaviors that would be undesirable in power or bandwidth constrained environments. Finally, our evaluation focuses primarily on single core workloads. Multi-core interference and shared cache effects may significantly alter the trade-offs explored in this work.

Despite these limitations, the observed trends are consistent across cache sizes and workloads, suggesting that the core conclusions regarding robustness and adaptability remain valid.

XI. Conclusion

This work explores the feasibility and effectiveness of using online machine learning to adaptively control hardware prefetching within a realistic microarchitectural simulation framework. We implemented an ML-based prefetch controller in gem5 that dynamically selects among multiple heterogeneous prefetchers using a lightweight contextual bandit formulation. Unlike prior work that focuses on designing increasingly complex prefetch algorithms, our approach treats prefetching as a runtime control problem, allowing the system to adapt its behavior based on observed execution feedback.

Across a broad evaluation using MachSuite benchmarks and multiple cache configurations, the ML-based controller consistently demonstrated robust performance. Although it did not universally outperform the single best static prefetcher for each workload, it reliably avoided catastrophic slowdowns and frequently matched or closely approached the best performing static policy. In most cases, the controller achieved second-best performance while rarely being the worst option, validating its effectiveness as a general purpose adaptive mechanism in the absence of application specific tuning.

Our results also highlight the inherent difficulty of prefetch control. Prefetch effectiveness depends on complex interactions between access patterns, cache capacity, memory-level parallelism, and contention for shared resources. As a result, traditional metrics such as prefetch accuracy do not always correlate with end to end performance. The ML controller prioritizes IPC-based reward rather than accuracy, which explains its tendency toward moderate accuracy yet competitive performance. This behavior reflects a deliberate design choice that favors performance robustness over aggressive specialization.

Importantly, our findings suggest that online learning is best suited for adaptability rather than absolute optimality. Even when trained and frozen on a given workload, the controller does not consistently exceed the best static prefetcher, emphasizing that lightweight bandit-based learning trades peak performance for stability and responsiveness. This outcome is consistent with the controller's intentionally minimal state representation and conservative exploration strategy, both of which are designed to maintain low overhead and hardware realism.

Overall, this work demonstrates that machine learning can be effectively integrated into prefetch control without resorting to heavyweight models or extensive offline profiling. While there remains significant room for improvement such as incorporating richer state features, phase detection, memory concurrency awareness, and multi objective reward functions our controller provides a strong foundation for future research. By framing prefetch selection as an online decision making problem, this work opens the door to more adaptive and resilient memory systems capable of operating efficiently across diverse workloads and architectural constraints.

XII. Future Work

There are several promising directions for extending this work to further improve the effectiveness and applicability of ML-based prefetch control. A key limitation of the current design is its reliance on IPC centric reward signals that do not explicitly capture memory system contention. Future versions of the controller could incorporate MSHR aware or memory-concurrency-sensitive reward shaping, similar in spirit to prior work such as APAC, to better distinguish between overlapped and exposed memory stalls. Integrating such signals would allow the controller to penalize aggressive prefetchers when they saturate memory resources and to favor more conservative policies under high contention.

Another important extension is the incorporation of lightweight phase detection. Many workloads exhibit distinct execution phases with dramatically different memory behaviors, and a single policy may not be optimal across all phases. By augmenting the state representation with phase indicators such as reuse distance trends, miss rate deltas, or simple PC based signatures the controller could adapt more quickly to behavioral shifts and reduce the lag inherent in epoch-based learning. This would be particularly beneficial for workloads with alternating streaming and irregular access patterns.

The current controller selects exactly one prefetcher per epoch, but future work could explore cooperative or blended prefetching strategies. Instead of enforcing mutual exclusivity, the controller could dynamically enable multiple prefetchers with bounded aggressiveness or selectively gate their output based on runtime feedback. Such hybrid strategies may offer improved robustness by combining the strengths of different prefetchers while limiting their individual drawbacks.

Expanding the controller's state representation is another promising avenue. While the present implementation intentionally uses a minimal state for hardware realism, incorporating additional microarchitectural features such as cache occupancy, bandwidth utilization, or prefetch queue pressure could significantly improve decision quality. These features would allow the learning agent to better reason about secondary effects such as cache pollution and memory backpressure, which are not fully captured by aggregate IPC alone.

From a learning perspective, future work could investigate alternative online learning algorithms beyond contextual bandits. Hierarchical or meta-learning approaches could enable faster adaptation across workloads, while transfer-learning mechanisms could allow policies learned on one application domain to generalize more effectively to related workloads. Additionally, adaptive exploration schedules that reduce exploration during stable phases and increase it during detected transitions may improve convergence speed without sacrificing robustness.

Finally, a comprehensive hardware feasibility analysis is necessary to assess the practicality of deploying this approach in real processor designs. Although the current controller is intentionally lightweight, future work should quantify area, power, and timing overheads, as well as integration complexity

with existing cache controllers. Evaluating these trade offs will be critical to determining whether ML-based prefetch control can be realistically adopted in production systems and what design constraints must be satisfied for successful deployment.

XIII. Team Responsibility

M Sadman Sakib was primarily responsible for the design and implementation of the machine learning based prefetch controller. This included formulating the contextual bandit problem, defining the state representation and reward function, and implementing the controller as a custom prefetcher within the gem5 simulator. He also developed mechanisms for per-epoch statistics collection, Q-table management, and optional training/freezing workflows. In addition, he conducted extensive initial testing, parameter tuning, and microbenchmark development to validate the controller's behavior across different cache configurations.

Munasib Ilham led the benchmarking and experimental evaluation effort. His contributions included setting up MachSuite benchmarks, configuring simulation scripts for different cache and prefetcher configurations, running large batches of gem5 simulations, and collecting and organizing performance statistics. He also performed baseline evaluations of static prefetchers and assisted in generating plots and tables used for comparative analysis.

Both members collaborated closely on debugging, result analysis, and interpretation, including identifying performance anomalies, explaining accuracy and coverage trends, and refining the experimental methodology. The final analysis and conclusions reflect joint discussions and iterative refinement by both team members.

Acknowledgment

The authors would like to thank Prof. Matthew D. Sinclair for their guidance, valuable suggestions, and continuous support throughout the course of this project.

References

[1] J. Jiménez et al., "Adaptive Prefetching on POWER7: Improving Performance and Power Consumption" IBM Research, 2014.

[2] J. Hiebel and L. E. Brown, "Machine Learning for Fine-Grained Hardware Prefetcher Control," Proc. ICPP, 2019.

[3] D. Glassner and K. Crammer, "Bandits Meet Computer Architecture: Designing a Smartly-allocated Cache," 2016.

[4] X. Lu, R. Wang, and X.-H. Sun, "APAC: An Accurate and Adaptive Prefetch Framework with Concurrent Memory Access Analysis," Proc. IISWC, 2020.

[5] H.-J. Yang, J. Fang, M. Cai, and Z. Cai, "A Prefetch-Adaptive Intelligent Cache Replacement Policy Based on Machine Learning," Journal of Computational Science and Technology, vol. 38, no. 2, 2023.

[6] N. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," Proc. HPCA, 2007.

[7] S. Ebrahimi, C. J. Lee, and Y. N. Patt, "Coordinated Control of Multiple Prefetchers in Multicore Systems," MICRO, 2009.

[8] B. Reagen, R. Adolf, Y. S. Shao, G. -Y. Wei and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," IEEE International Symposium on Workload Characterization (IISWC), 2014.

[9] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.