Extended Abstract

Hardware prefetching is a prediction problem: the processor tries to bring data into cache before the program asks for it. Pythia frames this as an online reinforcement learning problem, where the prefetcher observes hardware state, chooses a memory offset to prefetch or chooses no prefetch, and updates a small table of Q-values from reward feedback Bera et al. (2021). The original Pythia reward already captures whether a prefetch was accurate, timely, incorrect, out-of-bounds, or skipped, and it changes some rewards when DRAM bandwidth is high. However, it does not directly account for two costs that become important when the memory system is busy: a prefetch can evict useful cache data, and it can create a DRAM row-buffer conflict. Our project asks whether making Pythia's reward more system-aware can reduce harmful prefetches without breaking the parts of Pythia that already work well.

We made two main reward changes. First, we implemented a cache-pollution reward. Our initial direct version penalized any Pythia prefetch that evicted a valid line which had been used. This was too broad, so our final version, CPR-v2, keeps a small "badness" table indexed by a hash of the Pythia state and action. Pollution-like events increase badness for that state-action signature, useful prefetches decrease it, and future prefetches from high-badness signatures receive an extra penalty. This makes the reward target repeated bad patterns instead of taxing every suspicious eviction. Second, we implemented a row-buffer reward. This reward penalizes only the case where a Pythia prefetch was both useless and a DRAM row-buffer miss. We intentionally do not punish accurate prefetches that open a new row, because those prefetches still served useful data.

Our single-core experiments were an important negative result. On 16 SPEC traces, the row-buffer reward with penalty -4 reduced useless prefetches and row misses on some workloads, but the geomean IPC ratio was 0.99964 versus stock Pythia, essentially neutral. CPR-v2 also behaved more safely than the direct cache-pollution reward and reduced useless prefetches on some traces, but did not produce a consistent IPC improvement. This matched our intuition after the poster: on one core, reducing wasted prefetch traffic does not always free a real bottleneck.

The project became more interesting in multi-core experiments. We evaluated 16 homogeneous SPEC workloads at 4 cores and 8 cores, with each trace replicated across all cores to create high contention. Under this pressure, the same reward signals made more sense. Row-buffer shaping improved memory-bound traces more clearly at 8 cores: at the strongest tested row-buffer penalty, omnetpp improved by 1.7%, milc by 2.4%, and mcf by 1.5%, while useless prefetches and row misses often dropped by 10–20%. CPR-v2 also showed positive cases, including soplex and milc, though it remained more strength-sensitive. Combining both rewards improved 10 of 16 traces at both 4 and 8 cores, with best cases of +3.4% on milc at 4 cores and +2.6% on soplex at 8 cores. The combined reward did not reliably add the two benefits together, which suggests both mechanisms often suppress the same wasteful prefetches.

We also explored an offline warm-start direction using deep Conservative Q-Learning Kumar et al. (2020). The goal was to train a model from logged Pythia transitions and distill it back into Pythia's hardware-friendly table representation so online learning would not start from scratch. We hoped that this would improve early prefetches as well as allow Pythia's tabular sarsa to learn the behavior of richer deep models. We experienced limited success on this path. In general, warm-starting produced small same-trace IPC gains, but on fresh heldout traces the gains mostly flattened or turned negative.

Overall, our main contribution is an implementation and evaluation of two system-aware reward signals for Pythia. The most important lesson is not that one fixed reward setting always wins, but that reward shaping becomes more useful when the simulator has real shared-memory pressure. Future work should study adaptive reward weights or mixed workloads, because a fixed penalty helps memory-bound programs but can slightly hurt others.

System-Aware Reward Shaping for the Pythia RL Prefetcher

Esmee Cowing
Department of Computer Science
Stanford University
ecowing@stanford.edu

Tesvara Jiang
Department of Computer Science
Stanford University
tesvaraj@stanford.edu

Milly Wong
Department of Computer Science
Stanford University
millywy@stanford.edu

Abstract

Pythia is an online RL hardware prefetcher that learns a prefetching policy while a program runs. We extend Pythia with two system-aware reward signals: a cache-pollution badness reward and a row-buffer conflict reward. On single-core workloads, both rewards are mostly IPC-neutral even when they reduce useless prefetches. On 4-core and 8-core homogeneous workloads, where shared memory pressure is higher, the rewards produce clearer benefits on memory-bound traces. Our results suggest that reward shaping is most useful when the extra reward signal corresponds to a real bottleneck in the memory system. We also attempted to improve Pythia's performance using an offline warm start that utilized deep learning, though, on fresh heldout traces, we experienced limited success.

1 Introduction

Modern processors can execute much faster than main memory can respond. Hardware prefetchers try to hide this gap by predicting future memory accesses and fetching cache lines before they are requested. A good prefetch helps performance, but a bad prefetch can waste bandwidth, evict useful cache lines, or interfere with other memory requests.

Pythia treats prefetching as a contextual reinforcement learning problem Bera et al. (2021). The agent observes hardware-visible state, chooses a prefetch offset or no prefetch, and updates Q-values based on the outcome. This is appealing because it lets the prefetcher adapt to different programs. However, Pythia's original reward mostly focuses on prefetch usefulness, timeliness, and bandwidth level. Our project studies whether Pythia can make better decisions if its reward directly includes more system-level costs.

At the milestone, we had reproduced the Pythia baseline and run leave-one-out reward ablations. That helped us understand which original reward components mattered. After TA feedback, we narrowed the project away from broad algorithm swapping and focused on reward design. At the poster stage, our main reward results were still single-core and mostly neutral. Since then, the biggest progress has been running multi-core reward sweeps. Those results are more convincing because cache pollution and row-buffer conflicts are more harmful when multiple cores share memory bandwidth.

2 Related Work

Pythia is the main system we build on. It uses an online SARSA-style update with a compact Q-value store designed to be hardware-friendly Bera et al. (2021). Our work keeps Pythia's learning algorithm and table structure fixed, but changes the reward signals.

There is also broader work on learning-based prefetching and memory-system RL. Prior supervised approaches learn memory access patterns from program history Hashemi et al. (2018). Offline RL has also been used for related cache-management problems, where the model must learn from logged behavior instead of direct online interaction Kumar et al. (2020).

3 Background: Pythia

Pythia runs inside ChampSim, a trace-driven simulator. On each demand memory access, Pythia builds a state from features such as the load PC, page offset, recent address deltas, bandwidth level, and cache accuracy level. It then chooses an action from a small set of prefetch offsets plus a no-prefetch action. Pythia tracks the prefetch and later assigns a reward when the line is used, arrives late, is never used, goes out of bounds, or collides with an existing tracker entry.

The baseline reward has several components: accurate and timely prefetches are rewarded most, accurate but late prefetches receive a smaller reward, incorrect prefetches and out-of-bounds predictions are penalized, and no-prefetch receives a small penalty. Some rewards are split by low versus high bandwidth. Learning uses a SARSA-style update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[r_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right].$$

4 Method

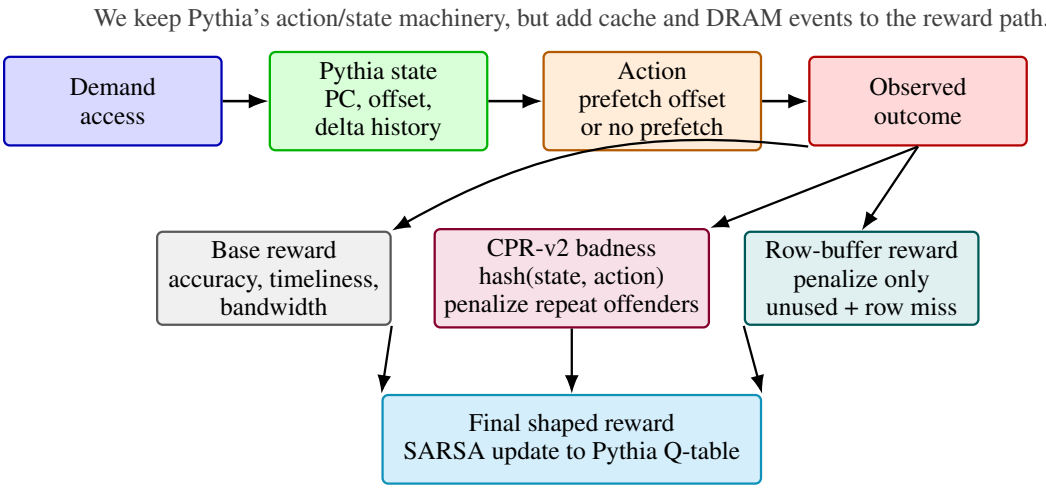

Figure 1: Pythia's online learning loop with our added reward-shaping paths. The base Pythia reward still handles accuracy and timeliness; CPR-v2 and row-buffer shaping add system-level feedback from cache and DRAM events.

4.1 Cache-Pollution Reward

Our first cache-pollution reward directly penalized a Pythia prefetch when its fill evicted a valid line that had already been used. This was easy to implement, but it fired too often. In practice, it behaved like a broad tax on prefetching rather than a targeted signal.

The final version, CPR-v2, uses a small badness table. For each Pythia prediction, we hash the state and action into a signature σ. Pollution-like events increase badness for σ: evicting a used victim is

Table 1: Reward variants evaluated in the final reward sweeps.

Variant family	What it discourages	Settings
Row-buffer reward	Useless prefetch plus DRAM row miss	off, -2, -4, -8
CPR-v2	Repeated polluting state-action patterns	off, p1, p2, p4
Combined	Both reward signals together	off, row-buffer -4 + CPR-v2 p1

a weak signal, evicting a prefetched line that dies unused is stronger, and later demand reuse of an evicted victim is strongest. Useful prefetches decrease badness. Then, when Pythia later computes reward for the same kind of state-action pattern, high badness adds a penalty:

$$R_{\text{cp}}(S, A) = R(S, A) + \rho(b(\sigma)),$$

where ρ becomes more negative as badness rises. This design avoids directly double-counting the event itself. Instead, the event changes future learning for repeated patterns.

4.2 Row-Buffer Reward

DRAM serves requests faster when they hit the currently open row. A request to a different row causes a row-buffer miss and can delay other traffic. We added instrumentation so a Pythia prefetch fill records whether its DRAM access was a row hit, row miss, or unknown. The reward shaping is deliberately narrow:

$$R_{\text{rb}}(S, A) = R(S, A) + \lambda_{\text{rb}} \mathbb{1}[A \text{ prefetched} \wedge \text{unused} \wedge \text{row miss}],$$

with $\lambda_{\text{rb}} < 0$. We only apply this when the prefetch is already classified as incorrect. If a prefetch is useful, we do not penalize it even if it opened a new DRAM row, because it still helped satisfy a real demand access.

4.3 Warm-Start

The warm-start direction logs Pythia transitions, trains an offline Q-function with Conservative Q-Learning, and distills the learned values back into Pythia's featurewise table representation. At the poster stage, transformer CQL was the strongest development-set candidate, but those runs overlapped model selection and evaluation traces, so we should not claim a final held-out win yet. This section should be filled once the group decides which warm-start model, if any, to report.

5 Experimental Setup

We use ChampSim with Pythia/Scooby at the L2 cache. The main metric is IPC, reported as system IPC for multi-core runs and as a percent change relative to stock Pythia. We also report useless prefetches and row-buffer misses when they explain the IPC result.

For reward tuning, we used 16 SPEC CPU traces. The single-core runs used 100M warmup instructions and 300M simulated instructions. The multi-core runs used 50M warmup and 250M simulated instructions per core. For 4-core and 8-core experiments, each trace was replicated across all cores. These homogeneous bundles are not a perfect model of real mixed workloads, but they intentionally create strong contention so we can test whether the reward matters when memory is busy.

6 Results

6.1 Single-Core Results

The single-core results were mostly neutral. Row-buffer reward with $\lambda_{\text{rb}} = -4$ had a geomean IPC ratio of 0.99964 versus stock Pythia across the 16 training traces. It improved milc by 0.25% and xalancbmk by 0.38%, but hurt mcf_s by 0.87%. It often reduced useless prefetches and row misses, but that did not automatically turn into IPC speedup. Our interpretation is that one core often has enough spare bandwidth that cutting bad prefetches does not free the real limiting resource.

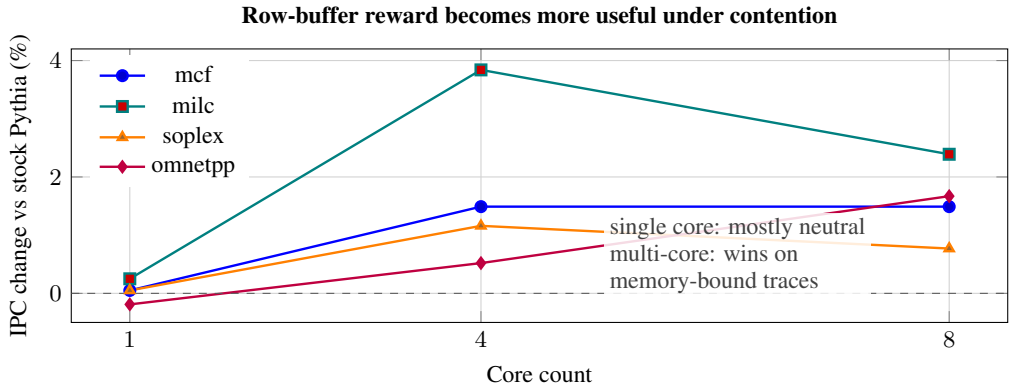

Figure 2: Representative row-buffer reward speedups as core count increases. The single-core results are close to neutral, while several memory-bound traces improve more clearly when multiple cores share the memory system.

Table 2: Representative multi-core results. These are not averages; they are selected examples that show where each reward is useful.

Reward	Cores	Trace	Best IPC change	Main effect
Row-buffer	4	milc	+3.8%	fewer useless prefetches and row misses
Row-buffer	8	omnetpp	+1.7%	useless prefetches down about 20%
CPR-v2	4	soplex	+3.0%	better behavior under noisy memory traffic
CPR-v2	8	milc	+2.8%	strength-sensitive but positive
Combined	8	soplex	+2.6%	both penalties active

CPR-v2 also looked safer than the first direct cache-pollution reward. On small development traces, it was IPC-neutral and more selective. For example, on gcc, stronger CPR-v2 settings reduced useless prefetches, while bwaves was almost untouched because baseline Pythia was already highly accurate. On mcf, CPR-v2 detected many pollution-like events but did not strongly change the policy. This was a useful implementation result, but not enough for a final performance claim by itself.

6.2 Multi-Core Results

The multi-core experiments show the clearest progress since the poster. With four or eight cores, a bad prefetch can consume bandwidth and row-buffer locality that other cores need. This is where the new reward signals make more sense.

For row-buffer reward, 4-core results were still often close to neutral, but the positive cases became larger: milc improved up to 3.8%, and soplex up to 1.2%. At 8 cores, the benefit grew with contention. With the strongest tested row-buffer penalty, omnetpp improved by 1.7%, milc by 2.4%, and mcf by 1.5%, while useless prefetches and row misses often dropped by 10–20%. Streaming and compute-bound traces were mostly flat, which is expected because they either already prefetch well or are not bottlenecked by this kind of memory waste.

For CPR-v2, the best cases were also memory-bound traces. At 4 cores, soplex improved by 3.0% with v2_p2, and milc improved by 2.2% with v2_p4. At 8 cores, soplex improved by 2.8% with v2_p1 and milc by 2.8% with v2_p4. However, CPR-v2 is more strength-sensitive than the row-buffer reward. Some settings that help one trace can hurt another, so v2_p1 is the safer default.

Combining the two rewards improved 10 of 16 traces at both 4 cores and 8 cores. The best combined cases were milc at 4 cores (+3.4%) and soplex at 8 cores (+2.6%). However, the combined reward did not consistently outperform the best single reward. This suggests that both rewards often target the same bad behavior: prefetches that are wasteful under contention.

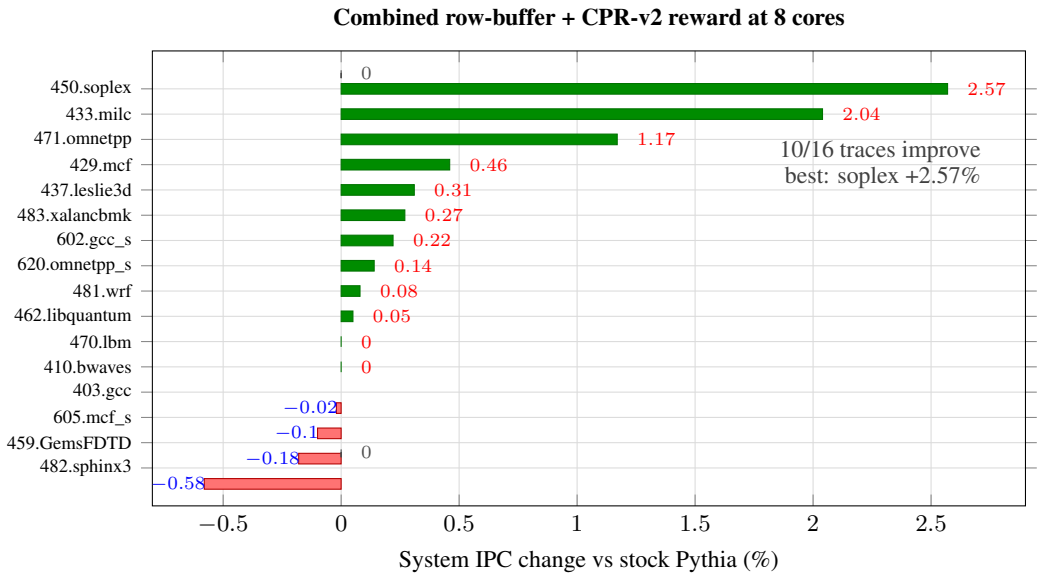

Figure 3: Per-trace 8-core IPC change for the combined reward setting. The combined reward improves 10 of 16 traces, with the largest gains on memory-bound workloads such as soplex, milc, and omnetpp.

6.3 Warm-Start Results

7 Discussion

The main lesson is that reward shaping does not create speedup just because it reduces a bad counter. On single-core runs, row misses and useless prefetches can fall while IPC stays the same. The reward matters more when the counter is tied to a real bottleneck. Multi-core contention makes that bottleneck visible: useless prefetches compete with other cores, and row-buffer conflicts slow down shared DRAM service.

The results are still workload-dependent. Fixed penalties are too blunt: the same strength that helps milc or soplex may slightly hurt xalancbmk or leslie3d. This points toward adaptive reward weights, per-workload tuning, or a learned mechanism that changes penalty strength based on observed contention.

Another caveat is that our multi-core workloads are homogeneous, meaning every core runs the same trace. This is useful as a stress test, but real systems run mixed workloads. A stronger final version would test mixed bundles where one noisy prefetcher can hurt another application's memory stream.

8 Conclusion

We implemented and evaluated two system-aware reward signals for Pythia: CPR-v2 for cache pollution and a row-buffer conflict penalty for useless prefetches. Single-core results were mostly neutral, which helped explain why the poster results did not look strong. Multi-core experiments were more positive because the new rewards corresponded to actual shared-resource pressure. Our best conclusion is that Pythia's reward can be improved, but the improvement is conditional: system-aware rewards help most when the system signal is a real bottleneck.

9 Team Contributions

Milly Wong set up much of the project workflow and experimental environment, including trace organization, baseline reproduction, and debugging the build/run setup. Milly also helped shape the

final report narrative around why the single-core result was not the full story and why multi-core contention was the right next evaluation.

Tesvara Jiang led the reward-design direction. Tesvara ran the leave-one-out ablations, implemented and tuned the row-buffer and cache-pollution reward mechanisms, added the multi-core Modal sweeps and summarizers, and wrote the multi-core findings that form the main result section.

Esmee Cowing led the offline RL and warm-start direction, including Conservative Q-Learning experiments, transformer/MLP warm-start candidates, data-collection coverage experiments, and analysis of why the current warm-start results should be treated cautiously until final model selection is finished.

10 AI Tools Disclosure.

We used Claude Code CLI (`claude-sonnet-4-6`, Anthropic) as a coding aid for helping set up project infrastructure. Specifically, Claude helped set up the Modal run pipeline for launching single- and multi-core sweeps, helped with data loading scripts, helped write the summarizing and analysis scripts that aggregate per-trace IPC and prefetch statistics, and assisted with general environment and build setup. Claude was also used for minor debugging throughout.

The essential reward-function and RL components of the project were designed and implemented independently by the team. The report prose was written by the team.

Changes from Proposal. Our original proposal split work across reward signals, RL algorithm comparisons, and exploration strategies. After TA feedback, we narrowed the main research contribution to reward design because simply swapping standard RL algorithms was less compelling. The warm-start work remained as a focused deep-RL component, but the main final-report results shifted toward reward shaping and multi-core evaluation. This adjustment was necessary because the research process showed that the reward signals were the part where we could make a clear systems contribution and explain both positive and negative results.

References

Rahul Bera, Konstantinos Kanellopoulos, Anant V. Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1121–1137. doi:10.1145/3466752.3480114

Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning Memory Access Patterns. In *Proceedings of the 35th International Conference on Machine Learning*.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative Q-Learning for Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems*.