COALESCE

Coherence-**O**bservant **A**daptive **LE**arning for **S**ystem-wide **C**ache **E**fficiency

Harsh Raj

February 14, 2026

Department of Computer Science
B.Tech Project Mid-Term Review

Literature Review & Research Gaps

Paper 1: Multiperspective Reuse (MICRO 2016)

Core Idea:

- Hashed perceptron combining multiple reuse signals
- 1-cycle prediction latency
- Optimized for hit-rate improvement

Key Limitation:

- Assumes all cache misses have equal cost
- No modeling of coherence traffic
- Single-core centric optimization

Our Extension

- Preserve hashing efficiency
- Inject MESI state + Sharer count
- Shift objective from hit-rate to system latency

Paper 2: Cost-Effective RL (HPCA 2021)

Core Idea:

- RL-based replacement under tight hardware budget
- Removes Program Counter to stay under 5KB

Key Limitation:

- Loss of spatial identity
- Cannot distinguish synchronization data from bulk arrays
- Reduced behavioral granularity

Our Extension

- Retain PC within 5KB budget
- Use 3% set sampling for update efficiency
- Bloom-filter ghost buffer for compact feedback tracking

Paper 3: RL for Multicore (Souza & Freitas, 2024)

Core Idea:

- Incorporates Sharer Count + Coherence State
- Reduces directory traffic vs blind LRU

Key Limitation:

- Operates as wrapper over LRU
- Victim restricted to LRU candidate
- Coarse sharer categorization

Our Extension

- Replace legacy policy entirely
- Native dual-hashed perceptron
- Score every line in set using exact sharer counts

Paper 4: Concurrency-Aware Miss Cost (2025)

Core Idea:

- Perceptron predicts variable miss latency
- Targets CPU stall reduction

Key Limitation:

- Focus on read-miss stalls
- Ignores dirty eviction overhead
- No system-level coherence cost modeling

Our Extension

- Composite cost model:

$$\text{Penalty} = \text{Stall} + \text{Traffic} + \text{Writeback}$$

- Optimizes total system cost, not just stall cycles

Problem Statement

Problem Statement

Observation:

- In multi-core systems, performance is often limited by **interconnect congestion** rather than compute speed.
- Coherence traffic increasingly dominates memory latency.

Limitation of Traditional LLC Policies (LRU, SRRIP):

- Optimize only for temporal hit rate
- Ignore coherence state and sharing behavior

Resulting Inefficiencies:

- **DRAM Overload:** Evicting `MODIFIED` blocks forces costly write-backs
- **Interconnect Traffic:** Evicting highly-shared blocks triggers invalidations

Core Problem

Existing LLC policies cannot distinguish high-cost coherence blocks from low-cost clean blocks, leading to avoidable system-wide latency inflation.

Objectives of the Work

Objective 1: Coherence-Aware Feature Extraction

Goal: Design a feature representation that captures both reuse behavior and coherence cost.

Approach:

- Combine Program Counter (PC)
- L2 Sharer Count
- MESI Coherence State

Expected Outcome:

- Generate a behavioral fingerprint for each cache line
- Enable cost-sensitive eviction decisions

Objective 2: Lightweight Learning-Based Replacement

Goal: Develop an adaptive replacement policy using a hardware-efficient learning model.

Approach:

- Dual-table hashed perceptron predictor
- Online weight updates based on reuse feedback
- Higher protection for high-cost blocks
- Faster penalization for zero-reuse streaming patterns

Expected Outcome:

- Improved victim selection across all ways in a set
- Reduced system-wide latency compared to static heuristics

Objective 3: Hardware-Constrained Design

Goal: Ensure architectural feasibility within strict silicon area limits.

Design Constraints:

- Total storage budget < 5KB
- Minimal impact on critical path

Techniques Used:

- 6.25% set sampling for controlled updates
- Bloom filter–based feedback tracking
- Compact PC signature transport

Objective 4: Approximating Belady's Optimal

Goal: Quantify how close the online policy approaches theoretical optimal replacement.

Methodology:

- Compare against Belady's MIN (offline oracle)
- Measure regret gap between practical and optimal policy

Expected Outcome:

- Demonstrate sufficiency of the 3-feature vector
- Establish theoretical grounding for design decisions

Detailed Architectural Features

Implemented Algorithmic Design

1. **Dual Orthogonal Hash Tables**
    - Two independently indexed weight matrices
    - Distinct bit-shift topologies and prime multipliers
    - Mitigates destructive aliasing across feature space

2. **Dynamic Training Threshold (τ)**
    - Bounds weight magnitudes to prevent saturation
    - Weight updates triggered only when:
        - $|y| < \tau$ or
        - Prediction error occurs
    - Enables faster adaptation during workload phase shifts

Stability & Efficiency Mechanisms

3. Override Mechanism for Strong Negative Confidence

- If predictor output crosses a critical negative threshold (e.g., -80), eviction proceeds even for high-cost states
- Prevents retention of dead modified lines under streaming-write patterns

4. 6.25% Set Sampling

- Global updates restricted to sets where:

$$\texttt{index}\%32 == 0$$

- Reduces dynamic update energy by 97%
- Maintains predictive accuracy via distributed learning

5. Bloom Filter–Based Feedback Buffer

- Tracks recently evicted partial tags
- Detects premature evictions

Future Work: Hardware Integration Roadmap (ChampSim)

1. NoC Header Piggybacking

- Embed compressed 12-bit PC signature within unused flit header bits
- Transfers PC information from L1 to LLC
- No additional wire overhead

2. Directory Bitmask-Based Sharer Extraction

- Sharer count derived directly from directory mask
- Hardware population count:

$$\texttt{popcount(directory\_mask)}$$

- Eliminates need for additional sharer tracking storage

Progress Achieved So Far

Custom Event-Driven C++ Simulator

- **Modelling:** 4-Core, 8MB LLC, 16-Way.
- **Baselines:** LRU, SRRIP, SHiP, SDBP (State-of-the-Art).
- **Latency Model:** Hit (15), DRAM (200), Coherence Penalty (+100).

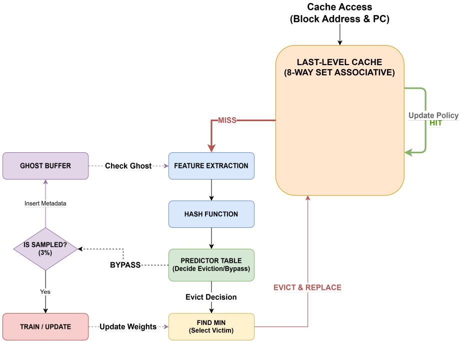


Figure 1: Simulation Flow

The Cost Function

1. The True Cost of a Miss:

Unlike standard policies, we account for system-wide traffic.

Penalty Formula

$$\text{Cost}(x) = \text{Stall} + \underbrace{\alpha \cdot \text{Traffic}}_{\text{Coherence}} + \underbrace{\beta \cdot \text{Energy}}_{\text{Writeback}}$$

2. The Perceptron Vote:

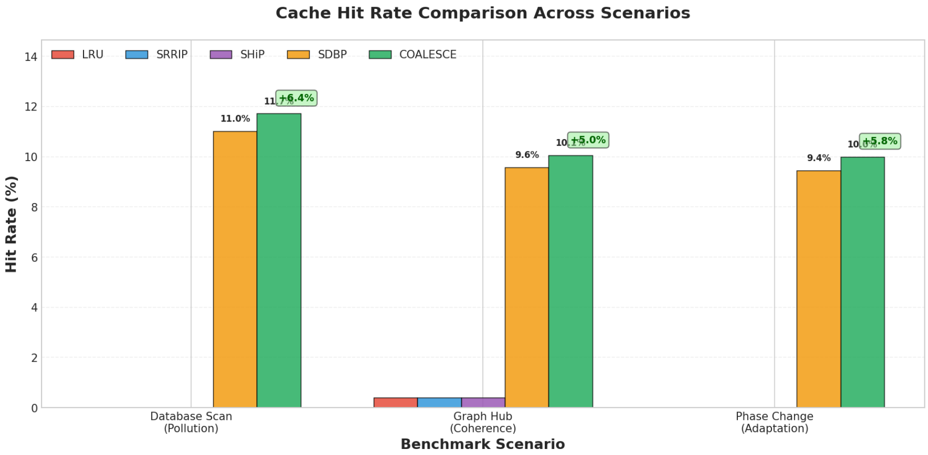
We use dual-hashing to avoid aliasing.

Vote Calculation

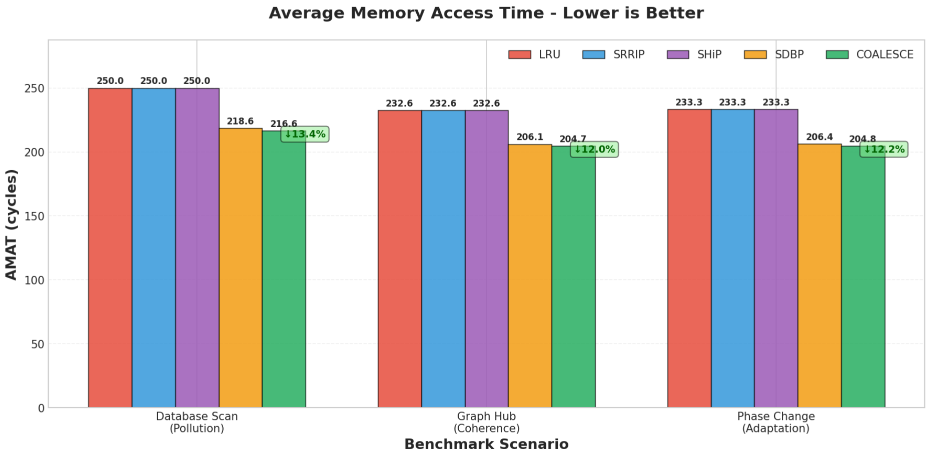
$$y = \sum_{i=0}^{N} w_i x_i + \text{Bias}_{\text{Coherence}}$$

$$\text{Bias} = \begin{cases} +150 & \text{if State} = \texttt{MODIFIED} \\ +75 & \text{if Sharers} \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

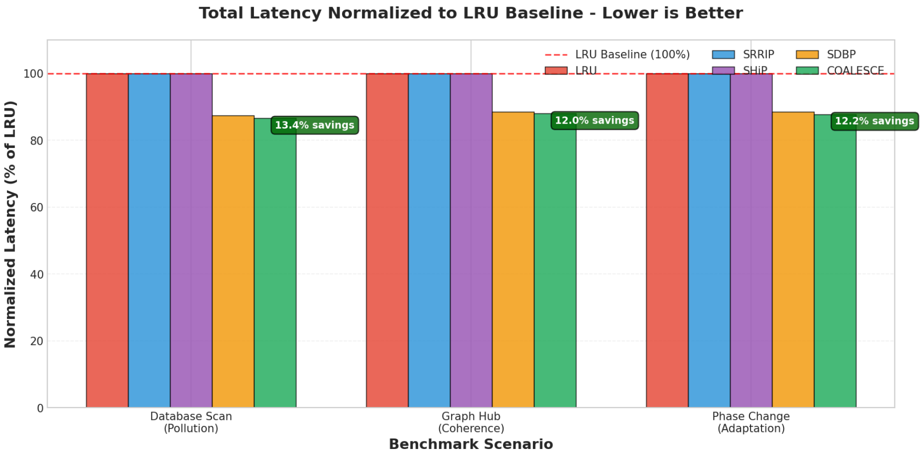
Result 1: Cache Hit Rate



Result 2: AMAT



Result 3: Latency



Result 4: Performance Summary

COALESCE Performance Summary

Scenario	LRU Hit%	SDBP Hit%	COALESCE Hit%	LRU AMAT	SDBP AMAT	COALESCE AMAT	Hit Rate Improve	AMAT Improve
Database Scan (Pollution)	0.00%	11.01%	11.72%	250.0	218.6	216.6	↑11.72%	↓13.4%
Graph Hub (Coherence)	0.41%	9.57%	10.05%	232.6	206.1	204.7	↑9.64%	↓12.0%
Phase Change (Adaptation)	0.00%	9.45%	10.00%	233.3	206.4	204.8	↑10.00%	↓12.2%

Future Work

Roadmap to Final Review

1. **Phase 2: ChampSim Integration**
   - Move from trace-based to cycle-accurate simulation.
   - Extract true IPC (Instructions Per Cycle) on **SPEC CPU 2017**.

2. **Theoretical Validation**
   - Train an **Offline RL Oracle** (Belady's Optimal).
   - Prove COALESCE feature vector is mathematically optimal.

Thank You