BTP Project Progress Report - 2

Project Progress Report: Advanced Architecture & Standalone Perceptron Simulation

Date: January 29, 2026

Author: Harsh Raj

Phase: Architecture Design & Advanced Prototyping (Phase 2)

Abstract

This week marked the transition from basic Q-Learning exploration to the rigorous system design of **Project COALESCE**. We finalized the architectural specifications for a **Hashed Perceptron-based Reinforcement Learning Cache Replacement Policy for Multi-Core Systems**.

To validate the feasibility of this complex logic within hardware constraints, we developed a sophisticated standalone simulator (`coalesce_sim.cpp`) incorporating **Adaptive Hashing**, **Ghost Buffers (Bloom Filters)**, and **Zero-Reuse Punishment**. Additionally, we conducted a comparative system design analysis of **LMCache** and **CacheBlend** to draw parallels between software-level KV-cache optimization and our hardware-level objectives.

---

1. Executive Summary

We have successfully locked in the final problem statement and architecture. The focus has shifted towards "Systems-First AI"—ensuring our RL agent is lightweight enough for a real Last-Level Cache (LLC).

Key Achievements:

- **Problem Statement Finalization:** Targeted design of a **Coherence-Aware Hashed Perceptron** policy.
- **Cross-Domain Analysis:** Studied **LMCache** and **CacheBlend** to understand state-of-the-art selective reuse mechanisms in distributed systems.

- **Advanced Simulation:** Built and verified a standalone C++ simulator implementing the full Perceptron + Ghost Buffer logic.

> ✓ **Key Milestone**
>
> The standalone simulator proved that our Perceptron logic can successfully identify and **bypass** "Scanner" workloads (streaming data) while protecting "Worker" workloads (shared data), maintaining a strict memory budget suitable for hardware integration.

---

2. Literature Review & System Design Analysis

Before finalizing the hardware logic, we analyzed two state-of-the-art software caching systems to understand how they handle "Reuse Prediction."

2.1 LMCache (KV Cache Middleware)

Concept: Enables sharing of KV caches across different GPU instances to reduce Time-To-First-Token (TTFT).

- **Relevance:** The problem of "Should I offload this tensor or keep it?" matches our "Evict or Keep?" decision in L3 cache.
- **Takeaway:** P2P sharing in LMCache validates our focus on **Sharer Counts**—data shared by multiple units is critical and must be protected.

2.2 CacheBlend (Selective Recomputation)

Concept: Solves the "Cross-Attention" bottleneck in RAG by recomputing only **High KV Deviation (HKVD)** tokens (approx. 15%).

- **Relevance:** CacheBlend demonstrates **Selective Intelligence**—we don't need to recompute everything, just the important bits.
- **Application to COALESCE:** Similarly, our Perceptron policy focuses on "Selective Caching." We identify the specific PCs causing cache pollution and filter them out.

---

3. Project COALESCE: The Architecture

We finalized the design of the **COALESCE Engine**. Unlike the Phase 1 Q-Table (which consumes too much RAM), this design uses **Hashed Perceptrons** to fit within a <5KB

hardware budget.

3.1 The "Brain" (Hashed Perceptron)

Instead of looking up a value in a massive table, the cache "votes" on a decision.

- **Inputs (Features):**
  1. **Program Counter (PC):** The instruction source.
  2. **Sharer Count:** The coherence state (from the Directory).
- **The Hash:** `Index = Hash(PC) ^ Hash(Sharer_Count)`
- **The Mechanism:** - A table of small 8-bit integers (Weights).
  - **Positive Weight:** Keep line (Important).
  - **Negative Weight:** Evict line (Useless).

3.2 The "Memory" (Ghost Buffer)

In RL, we need to know if an eviction was a *mistake*.

- **Implementation:** A **Bloom Filter** stores the hash of recently evicted lines.
- **Feedback Loop:** If a miss occurs for a line in the Ghost Buffer, we know we evicted it too early. We send a **negative reward** to update the weights.

3.3 Zero-Reuse Punishment

We identified a flaw in standard RL: "Scanners" (streaming data) never re-request data, so they never trigger a Ghost Buffer hit.

- **Fix:** We track if a line was *never touched* before eviction.
- **Logic:** If `Hits == 0` at eviction time → **Punish the PC immediately.**

---

4. Simulation Results (`coalesce_sim.cpp`)

We implemented the architecture in a standalone C++ simulator to stress-test the logic before ChampSim integration.

4.1 The Test Scenario

We created a synthetic adversarial workload:

1. **The Worker (PC `0xF00D`):** Accesses a small set of data repeatedly. **Sharer Count = 4** (High contention).

2. **The Scanner (PC `0xBAD`):** Accesses new addresses constantly (Streaming). **Sharer Count = 0** (Private).

4.2 The Results

The simulation ran for 10 epochs. The Perceptron successfully learned to distinguish the two patterns.

Simulation Output:

```
--- Starting COALESCE Standalone Sim ---
Config: Adaptive Hashing (2-Table), Ghost Buffer, Bypassing

Results:
Hits: 1999
Misses: 54
Bypasses: 1947

--- Brain Inspection ---
Vote for PC_WORK (0xF00D, Sharers=4): 254 (Should be Positive)
Vote for PC_SCAN (0xBAD,  Sharers=0): -92 (Should be Negative)
SUCCESS: The scanner is being bypassed!
```

ⓘ Analysis

- **PC `0xF00D` (Worker):** Weight saturated to **Positive (+254)**. Decision: **CACHE**.
- **PC `0xBAD` (Scanner):** Weight saturated to **Negative (0 or lower)**. Decision: **BYPASS**.
- **Result:** The "Scanner" data was bypassed, leaving the cache free for useful "Worker" data.

---

5. Next Steps: ChampSim Integration

With the logic verified, Phase 3 focuses on the "Real Systems" implementation.

5.1 Objectives for Next Week

1. Reading Research Paper on **Reinforcement Learning-Based Cache Replacement Policies for Multicore Processors** by MATHEUS A. SOUZA , (Member, IEEE), AND HENRIQUE C. FREITAS , (Member, IEEE)
2. **Core Modification:** Modify `inc/block.h` and `src/cache.cc` in ChampSim to plumb the `sharer_count` signal from the Directory to the Replacement Policy.

3. **Policy Porting:** Translate `coalesce_sim.cpp` into a ChampSim-compatible replacement module (`replacement/coalesce/coalesce.cc`).
4. **Benchmarking:** Run the new policy against the `perlbench` baseline established in Phase 1 to measure IPC improvement.
